UNISYS

A Series
DMS II Interpretive Interface
Programming Reference Manual

Copyright© 1987, Unisys Corporation
All Rights Reserved
Unisys is a trademark of Unisys Corporation
Previous Title: DMSII Interpretive Interface User's Manual

Relative to Release
Level Mark 3.7.0

Priced Item

July 1987
Distribution Code SE
Printed in U S America
1163813.370

UNISYS

A Series
DMS II Interpretive Interface
Programming Reference Manual

Copyright© 1987, Unisys Corporation
All Rights Reserved
Unisys is a trademark of Unisys Corporation
Previous Title: DMSII Interpretive Interface User's Manual

Relative to Release
Level Mark 3.7.0

Priced Item

July 1987
Distribution Code SE
Printed in U S America
1163813.370

The names, places, and/or events used in this publication are not i
correspond to any individual, group, or association existing, living,
Any similarity or likeness of the names, places, and/or events with
any individual living or otherwise, or that of any group or associatic
coincidental and unintentional.

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THE DOCU
product and related material disclosed herein are only furnished p
subject to the terms and conditions of a duly executed Program F
Agreement to purchase or lease equipment. The only warranties r
any, with respect to the products described in this document are
License or Agreement. Unisys cannot accept any financial or othe
may be the result of your use of the information in this document
material, including direct, indirect, special, or consequential dama

You should be very careful to ensure that the use of this informat
material complies with the laws, rules, and regulations of the juris
respect to which it is used.

The information contained herein is subject to change without nc
be issued to advise of such changes and/or additions.

Correspondence regarding this publication may be forwarded us
form at the back of the manual; submitted on a Field Communic
Class specified as 2 (System Software), the Type specified as 1 (
Product specified as the form number of this manual (1163813.
directly to Unisys Corporation, Software Documentation, 19 Mor
California, 92718, U.S.A.

JAN - 5 1995

UNISYS Publication Change Notice (PCN)

Date	Form—PCN number
May 28, 1989	1163813-003

Title: A Series DMSII Interpretive Interface Programming Reference Manual

Description: (Relative to the Mark 3.8.0 System Software Release)

This letter announces the availability of PCN-001 to the A Series DMSII Interpretive Interface Programming Reference Manual, form number 1163813, dated July 1987. This PCN contains changes to the error messages that can be returned by DMSII. This PCN also replaces references to the file DATABASE/INTERFACE with SYSTEM/DMINTERFACE. All revisions and additions are relative to the Mark 3.8.0 System Software Release.

Changes to the text are indicated by vertical black lines on the affected pages.

Replace These Pages	Add These Pages
v	Page Status Page
13	364A
19 through 21	372A
363	436A
371	446A
435	450A
445	
449	

Retain the PCN cover sheet as a record of changes made to the basic publication.

Copyright (c) 1989 Unisys Corporation
Unisys Corporation
All Rights Reserved

Printed in U S America

Page Status

Page	Issue
iii through iv	Original
v through vi	PCN-003
viii through x	Original
1	Original
2	Blank
3	Original
4	Blank
5 through 12	Original
13 through 14	PCN-003
15 through 18	Original
19 through 21	PCN-003
22	Blank
23 through 93	Original
94	Blank
95 through 179	Original
96	Blank
97 through 343	Original
344	Blank
345 through 361	Original
362	Blank
363 through 364A	PCN-003
364B	Blank
365 through 366	PCN-003
367 through 368	Original
369 through 372A	PCN-003
372B	Blank
373 through 374	PCN-003
375	Original
376	Blank
377 through 387	Original
388	Blank
389 through 393	Original
394	Blank
395 through 433	Original
434	Blank
435 through 436A	PCN-003
436B	Blank
437 through 444	Original
445 through 446A	PCN-003
446B	Blank
447 through 448	Original
449 through 450A	PCN-003
450B	Blank
451 through 466	Original

ABOUT THIS MANUAL

PURPOSE

This manual documents all DMSII Interpretive Interface functions, including commands, programs, statements, and status messages. It is to be used for referencing specific information.

SCOPE

This manual explains how to generate the DMINTERPRETER Library and how to write application programs that call entry points of the DMINTERPRETER Library.

AUDIENCE

This manual is written primarily for application programmers and secondarily for data base administrators.

PREREQUISITES

Anyone using this manual should be familiar with basic DMSII concepts.

HOW TO USE THIS DOCUMENT

Read the section "All About Interpretive Interface" for an overview of this product and for an explanation of concepts used in this manual. Read specific language sections for related entry points and their usage.

Within this manual, specific and possibly unfamiliar conventions are used, such as particular acronyms, railroad diagrams, and references to other manuals. The acronyms used in this manual are spelled out and defined in the glossary. An explanation of railroad diagrams can be found in the appendix "Understanding Railroad Diagrams." Manuals referred to in the text are for the A Series systems unless otherwise specified.

ORGANIZATION

Sections

1 ALL ABOUT INTERPRETIVE INTERFACE

Defines the interpretive interface and explains its unique capabilities and advantages for application program coding. Provides a data base definition that is used for all sample programs and sample program fragments throughout the manual.

2 GENERATING THE DMINTERPRETER LIBRARY

Discusses the files used to generate the DMINTERPRETER Library, and details three methods of generating it.

3 ENTRY POINTS IN COBOL/COBOL74

Describes entry points of the DMINTERPRETER Library in the COBOL and COBOL74 languages, and explains how to write user programs that call these entry points.

4 ENTRY POINTS IN ALGOL

Describes entry points of the DMINTERPRETER Library in the ALGOL language, and explains how to write user programs that call these entry points.

5 ENTRY POINTS IN PL/I

Describes entry points of the DMINTERPRETER Library in the PL/I language, and explains how to write user programs that call these entry points.

6 ENTRY POINTS IN FORTRAN77

Describes entry points of the DMINTERPRETER Library in the FORTRAN77 language, and explains how to write user programs that call these entry points.

7 DETERMINING THE DATA BASE STRUCTURE

Discusses the use of the data set named DBSTRUCTURE that allows application programs to determine the structure of the data base at run time. Lists and describes DBSTRUCTURE items.

Appendixes

A UNDERSTANDING RAILROAD DIAGRAMS

Explains how to read the syntax diagrams used in the manual.

B DMSII EXCEPTIONS AND ERRORS

Lists all exceptions and errors that can be returned by the Accessroutines when data management operations are attempted.

C SYNTAX DIAGRAMS FOR ENTRY POINT PARAMETERS

Describes the syntax of the parameters for all DMINTERPRETER entry points, including basic data base constructs.

D DBFIND ENHANCEMENTS

Explains how to use improvements that have been made to data base search algorithms.

E PROGRAMMING CONSIDERATIONS

Discusses recommendations for efficient and economical application programming with the interpretive interface.

F SAMPLE PROGRAMS

Lists sample programs in COBOL, COBOL74, ALGOL, PL/I, and FORTRAN77.

In addition, a glossary, a bibliography, and an index appear at the end of this manual.

RELATED PRODUCT INFORMATION

Title	Form No.
A Series ALGOL Programming Reference Manual, Volume 1: Basic Implementation	1169844
A Series COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation	1169877
A Series COBOL Programming Reference Manual	1169786
A Series DMSII User Language Interface Programming Guide	1180536
A Series DMSII Utilities Operations Guide	1163839
A Series FORTRAN77 Programming Reference Manual	1222666
A Series PL/I Reference Manual	1169620

CONTENTS

ABOUT THIS MANUAL iii

1 **ALL ABOUT INTERPRETIVE INTERFACE** 1
DMINTERPRETER LIBRARY. 5
DMINTERPRETER Entry Points 5
Standard Entry Points. 7
Data Transfer Entry Points 7
Exception-Handling Entry Points. 9
Attribute-Setting Entry Points 10
SAMPLE DATA BASE 11

2 **GENERATING THE DMINTERPRETER LIBRARY** 13
DMINTERPRETER FILES. 13
DMINTERPRETER GENERATION 15
Generation from Remote Terminals 15
Remote Generation Using the NOZIP Option 18
Generation from Job Decks. 19

3 **ENTRY POINTS IN COBOL/COBOL74.** 23
THE COBOL/COBOL74 INTERFACE TO DMINTERPRETER 25
INVOCATION OF ENTRY POINTS 27
3.1 STANDARD ENTRY POINTS. 29
DBOPEN 29
DBCLOSE. 31
DBBEGINTRANSACTION 32
DBENDTRANSACTION 34
DBCREATE 36
DBRECREATE 38
DBSTORE. 40
DBFREE 42
DBSET. 44
DBFIND 46
DBLOCK 49
DBDELETE 52
DBVERB 55
3.2 DATA TRANSFER ENTRY POINTS 60
DBGETDISPLAY 61
DBGETKANJI 63
DBGETREAL. 65
DBGETBOOLEAN 67
DBGETDOUBLE. 69
DBPUTDISPLAY 72
DBPUTKANJI 74
DBPUTREAL. 76
DBPUTBOOLEAN 78
DBPUTDOUBLE. 80
DBPUTNULL. 82
DBDATA 84
3.3 EXCEPTION-HANDLING ENTRY POINTS. 86

DBSTATUS 87
DBEXCEPTIONTEXT. 88
DBEXCEPTIONNAME. 90
3.4 ATTRIBUTE-SETTING ENTRY POINTS 92
DBSETLIMIT 92

4 ENTRY POINTS IN ALGOL. 95
LIBRARY DECLARATION. 97
ENTRY POINT DECLARATION. 99
INVOCATION OF ENTRY POINTS 102
4.1 STANDARD ENTRY POINTS. 103
ALGOLOPEN. 103
ALGOLCLOSE 105
ALGOLBEGINTRANSACTION. 106
ALGOLENDTRANSACTION. 108
ALGOLCREATE. 110
ALGOLRECREATE. 112
ALGOLSTORE 114
ALGOLFREE. 116
ALGOLSET 118
ALGOLFIND. 121
ALGOLLOCK. 125
ALGOLDELETE. 129
ALGOLVERB. 133
4.2 DATA TRANSFER ENTRY POINTS 138
ALGOLGETSTRING 139
ALGOLGETKANJI. 142
ALGOLGETREAL 144
ALGOLGETBOOLEAN. 147
ALGOLGETDOUBLE 150
ALGOLPUTSTRING 153
ALGOLPUTKANJI. 156
ALGOLPUTREAL 159
ALGOLPUTDOUBLE 162
ALGOLPUTBOOLEAN. 165
ALGOLPUTNULL 168
ALGOLDATA. 170
4.3 EXCEPTION-HANDLING ENTRY POINTS. 173
ALGOLSTATUS. 174
ALGOLEXCEPTIONTEXT 175
ALGOLEXCEPTIONNAME 177
4.4 ATTRIBUTE-SETTING ENTRY POINTS 179
ALGOLSETLIMIT. 179

5 ENTRY POINTS IN PL/I 181
LIBRARY DECLARATION. 183
ENTRY-POINT DECLARATION. 185
INVOCATION OF ENTRY POINTS 188
5.1 STANDARD ENTRY POINTS. 189
ALGOLOPEN. 189
ALGOLCLOSE 191
ALGOLBEGINTRANSACTION. 192
ALGOLENDTRANSACTION. 194

ALGOLCREATE. 196
ALGOLRECREATE. 198
ALGOLSTORE 200
ALGOLFREE. 202
ALGOLSET 204
ALGOLFIND. 206
ALGOLLOCK. 209
ALGOLDELETE. 213
ALGOLVERB. 216
5.2 DATA TRANSFER ENTRY POINTS 221
ALGOLGETSTRING 222
ALGOLGETKANJI. 225
ALGOLGETREAL 227
ALGOLGETBOOLEAN. 230
ALGOLGETDOUBLE 233
ALGOLPUTSTRING 236
ALGOLPUTKANJI. 239
ALGOLPUTREAL 242
ALGOLPUTDOUBLE 245
ALGOLPUTBOOLEAN. 248
ALGOLPUTNULL 251
ALGOLDATA. 253
5.3 EXCEPTION-HANDLING ENTRY POINTS. 255
ALGOLSTATUS. 256
ALGOLEXCEPTIONTEXT 257
ALGOLEXCEPTIONNAME 259
5.4 ATTRIBUTE-SETTING ENTRY POINTS 261
ALGOLSETLIMIT. 261

6 ENTRY POINTS IN FORTRAN77. 263
LIBRARY DECLARATION. 265
ENTRY POINT DECLARATION. 267
INVOCATION OF ENTRY POINTS 270
6.1 STANDARD ENTRY POINTS. 271
FORTRAN77OPEN. 271
FORTRAN77CLOSE 273
FORTRAN77BEGINTRANSACTION. 274
FORTRAN77ENDTRANSACTION. 276
FORTRAN77CREATE. 278
FORTRAN77RECREATE. 280
FORTRAN77STORE 282
FORTRAN77FREE. 284
FORTRAN77SET 286
FORTRAN77FIND. 288
FORTRAN77LOCK. 291
FORTRAN77DELETE. 294
FORTRAN77VERB. 297
6.2 DATA TRANSFER ENTRY POINTS 302
FORTRAN77GETCHARACTER. 303
FORTRAN77GETKANJI. 306
FORTRAN77GETREAL 308
FORTRAN77GETLOGICAL. 311
FORTRAN77GETDOUBLE 314

FORTRAN77PUTCHARACTER. 317
FORTRAN77PUTKANJI. 320
FORTRAN77PUTREAL 323
FORTRAN77PUTDOUBLE 326
FORTRAN77PUTLOGICAL. 329
FORTRAN77PUTNULL 332
FORTRAN77DATA. 334
6.3 EXCEPTION-HANDLING ENTRY POINTS. 337
FORTRAN77STATUS. 338
FORTRAN77EXCEPTIONTEXT 339
FORTRAN77EXCEPTIONNAME 341
6.4 ATTRIBUTE-SETTING ENTRY POINTS 343
FORTRAN77SETLIMIT. 343

7 DETERMINING THE DATA BASE STRUCTURE. 345
DBSTRUCTURE ITEMS. 348

A UNDERSTANDING RAILROAD DIAGRAMS. 353

B DMSII EXCEPTIONS AND ERRORS. 363

C SYNTAX DIAGRAMS FOR ENTRY POINT PARAMETERS 377

D DBFIND ENHANCEMENTS. 389

E PROGRAMMING CONSIDERATIONS 393

F SAMPLE PROGRAMS. 395

GLOSSARY. 435

BIBLIOGRAPHY. 450

INDEX 451

1 ALL ABOUT INTERPRETIVE INTERFACE

The main advantage offered by the Data Management System II (DMSII) Interpretive Interface is increased flexibility in writing application programs for data base processing. The interpretive interface allows dynamic run-time access to DMSII data bases, as opposed to static DMSII user language interfaces that are commonly used.

Traditionally, application programs have accessed DMSII data bases through the DMSII user language interface. This interface requires the use of specially modified user language compilers, which recognize DMSII constructs and translate them directly into calls on the data base. (Figure 1-1 shows the modules involved in compiling an application program for use with the user language interface.) Such an interface is rigid and static because all information about data base operations that is needed to execute the application program must be known when the program is compiled.

Although static interfaces allow optimal run-time performance, the dynamic interpretive interface offers the following advantages:

1. Application programs need not contain DMSII user language interface statements.

2. The data base does not have to be compiled at the time the application program is compiled. With the interpretive interface, decisions about data base operations can be postponed until run time.

3. Application programs written in languages that have not been extended to include DMSII user language statements, such as FORTRAN77, are nonetheless able to access data bases through the interpretive interface. (However, the application programs must include a library interface.)

4. Structural changes or new features in a data base do not affect application programs that utilize the interpretive interface, so there is little need for recompilation.

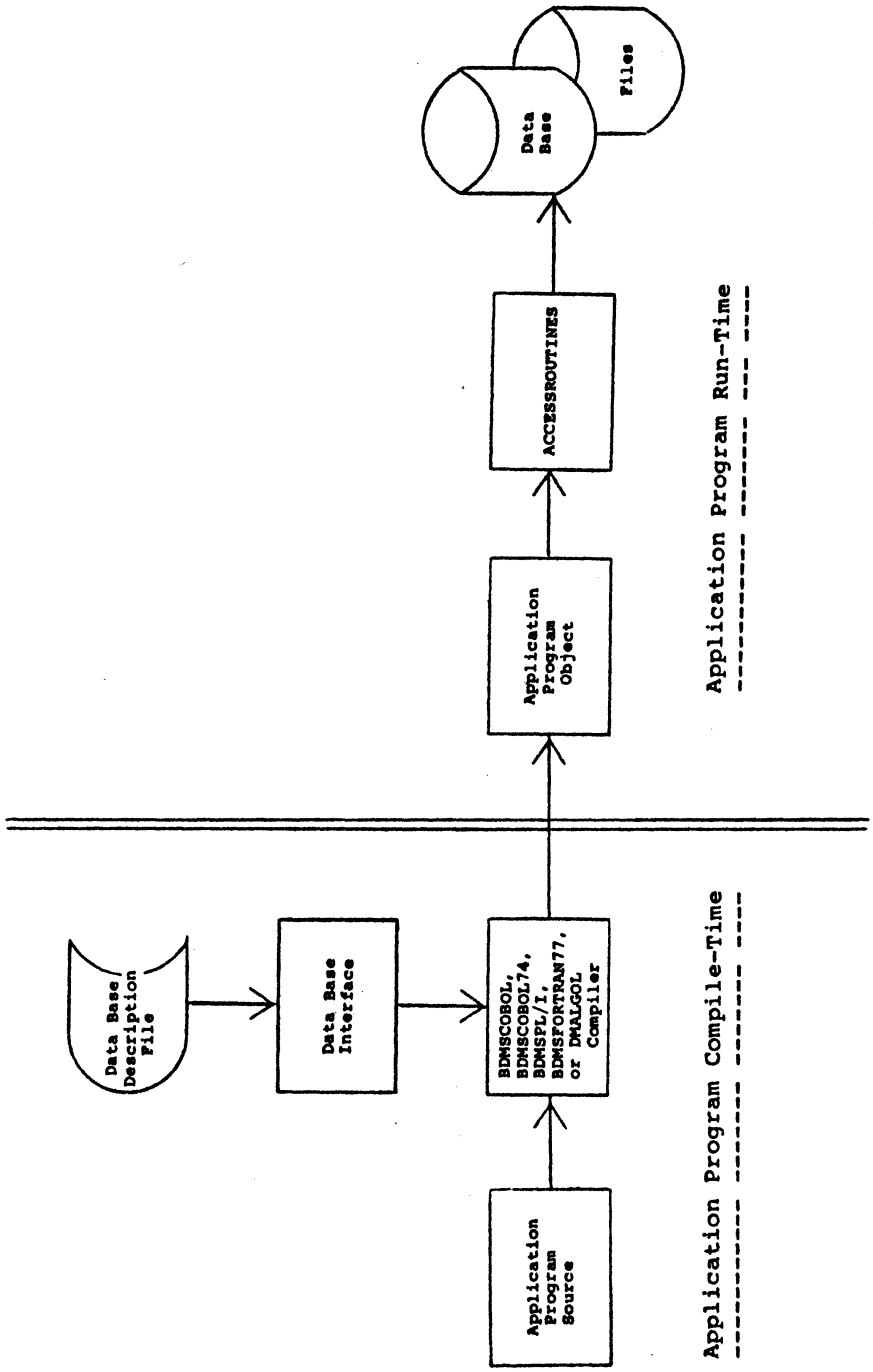


Figure 1-1. Data Base
Processing with User Language Interface

The main software component of the interpretive interface is the DMINTERPRETER Library. To access a data base using this library, an application program calls library entry points rather than executing DMSII user language statements. As Figure 1-2 demonstrates, it is possible to write and compile application programs before the data base is compiled, because this kind of application program never directly accesses the data base. Parameters of DMINTERPRETER entry points, which detail the types of operations to be performed by the entry points, need not be specified until the application program is run.

The interpretive interface utilizes another software component called the DMSUPPORT Library, which is a general support library used with all DMSII software. Because the DMSUPPORT Library contains error messages for all DMSII software, including the interpretive interface, compilation of the DMSUPPORT Library takes place during standard data base compilation. (For more information, refer to "Exception-Handling Entry Points" later in this section.)

Other facilities provided by the interpretive interface that are not provided by the user language interface include the following:

1. The intrinsic data set named DBSTRUCTURE that can be used to interrogate the structure of a data base at run time.

2. The parameterized OPEN statement allows the writing of general-purpose programs that can access any data base.

3. The library mechanism allows a single application program to invoke as many data bases as desired, provided that a DMINTERPRETER Library has been compiled for each data base.

Among the limitations of the interpretive interface is the lack of facilities for maintaining links and embedded manual subsets, which are other methods of accessing data in a data base. Nor does the interpretive interface provide counterparts to the DMSII user language statements known as INSERT, REMOVE, ASSIGN, and GENERATE.

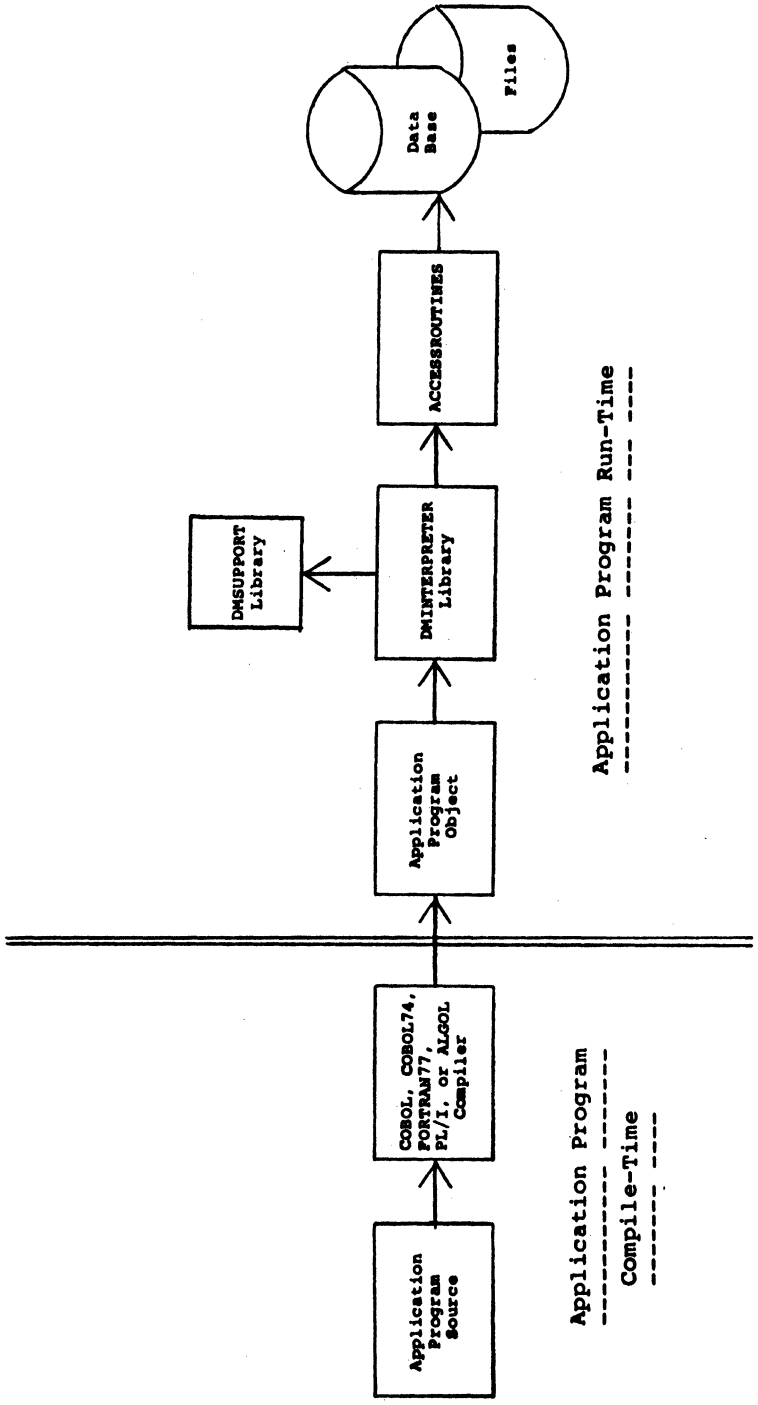


Figure 1-2. Data Base
Processing with the Interpretive Interface

DMINTERPRETER LIBRARY

The DMINTERPRETER library is written in DMALGOL and tailored to a particular data base during compilation. Entry points are provided that allow the library to be called from ALGOL, COBOL, COBOL74, FORTRAN77, and PL/I application programs. DATABASE/DMINTERPRETER, the library symbolic file, contains declarations of library entry points that can be incorporated into ALGOL and FORTRAN77 programs through the $ INCLUDE compiler control option. (The sequence ranges of declarations that can be included are discussed in a later section of this manual.) For PL/I, the $ INCLUDE compiler control option is not available, so each entry point declaration must be copied from the library symbolic file. The COBOL and COBOL74 languages do not require explicit library declaration.

The includable library declarations provide entry points that perform appropriate data-type transfers of the parameters and procedure results from each language. Subsets containing only some entry points can be declared, and entry points can be renamed. These procedures are further described in the sections of this manual that describe individual entry points for the various languages.

DMINTERPRETER Entry Points

There are four types of DMINTERPRETER entry points: standard, data transfer, exception handling, and attribute setting. These types are described in general terms in this section; more detailed discussions of entry points in ALGOL, PL/I, COBOL, COBOL74, and FORTRAN77 are provided in later sections.

Most DMINTERPRETER entry points perform functions that are similar to the functions of user language interface statements, and their names are formed by prefixing the name of the corresponding user language interface statement with "DB". For example, the DMINTERPRETER entry points known as DBFIND and DBOPEN correspond to the FIND and OPEN user language interface statements. Because library entry points cannot be renamed in COBOL and COBOL74, the "DB" prefixed entry points must be used for these two languages. The renaming facilities in ALGOL, FORTRAN77, and PL/I are used to produce "DB" entry points in each of these languages. Renaming is explained in detail in later sections of this manual.

Most entry points use the same number of parameters in the same order in all languages. The declared parameter types vary among the different languages, but all contain the same information. Basic syntax diagrams describing the internal format of the entry point .parameters are provided with individual entry point descriptions in each language. Refer to the appendix "Syntax Diagrams for Entry Point Parameters" for more detailed syntax diagrams for data base constructs used with the interpretive interface.

The syntax and semantics of the parameters for each entry point are very similar to those for the corresponding user language interface statements. Behavioral differences between DMINTERPRETER entry points and user language interface statements are described where appropriate.

All DMINTERPRETER entry points return an indication of an exception through the procedure result, unless otherwise noted for a particular language. The declared type of the result varies among languages. The interpretation of exceptions is further explained in this section under "Exception Handling Entry Points."

The interpretive interface is designed to provide a "record-at-a-time" interface to the data base, so that application programs should make use of key entry points in the following manner:

1. A record should be located using the DBFIND entry point, and the values of the items should be examined using the appropriate "DBGET-" entry points.

2. A new record should be added to the data base by first calling the DBCREATE entry point, then moving the values of each item into the user work area by calling the appropriate "DBPUT-" entry points, and finally storing the record by calling the DBSTORE entry point.

3. A record should be modified by first calling the DBLOCK entry point to locate it, then calling the appropriate "DBPUT-" entry points to alter the values of the items, and finally calling the DBSTORE entry point to store the modified values.

4. A record should be removed from the data base by calling the DBDELETE entry point.

Standard Entry Points

Standard entry points correspond to functions performed by DMSII user language interface statements. The results returned by standard entry points are used for exception handling.

Table 1-1 shows the exported names of standard entry points recognized by the DMINTERPRETER Library for each language. The names in this table can be used to declare DMINTERPRETER entry points in application programs, unless renaming is desired or necessary due to the limitations of a particular language.

Table 1-1. Standard Entry Points

COBOL and COBOL74	ALGOL and PL/I*	FORTRAN77
DBOPEN	ALGOLOPEN	FORTRAN77OPEN
DBCLOSE	ALGOLCLOSE	FORTRAN77CLOSE
DBBEGINTRANSACTION	ALGOLBEGINTRANSACTION	FORTRAN77BEGINTRANSACTION
DBENDTRANSACTION	ALGOLENDTRANSACTION	FORTRAN77ENDTRANSACTION
DBCREATE	ALGOLCREATE	FORTRAN77CREATE
DBSTORE	ALGOLSTORE	FORTRAN77STORE
DBFREE	ALGOLFREE	FORTRAN77FREE
DBSET	ALGOLSET	FORTRAN77SET
DBFIND	ALGOLFIND	FORTRAN77FIND
DBLOCK	ALGOLLOCK	FORTRAN77LOCK
DBDELETE	ALGOLDELETE	FORTRAN77DELETE
DBVERB	ALGOLVERB	FORTRAN77VERB

* PL/I directly calls the ALGOL entry points.

Data Transfer Entry Points

Data transfer entry points enable programs to transfer data from the data base to DMSII user work areas and back.

The DMSII user language interface makes user work areas directly accessible to application programs written in certain languages. This direct access is possible because the Data and Structure Definition Language (DASDL) data base description is known during compilation, and is used to enforce data type checking. However, the interpretive interface does not allow direct manipulation of DMSII user work areas because data type checking cannot be directly enforced, and the format of a user work area may not be known.

Instead, the DMINTERPRETER Library provides "get" and "put" entry points to move data to and from DMSII user work areas. These entry points allow data to be correctly moved in and out of a DMSII user work area without making the user work area directly accessible to the application program. "Get" entry points move data from the DMSII user work area to user-declared variables. "Put" entry points move data from user-declared variables to DMSII user work areas. Each item needed from a data set record must be individually moved, by calling the DMINTERPRETER Library, to a user-declared variable before it can be manipulated by the application program.

The DMINTERPRETER Library checks the parameters of data transfer entry points to ensure that the requested operation is compatible with the DASDL data base description. (For example, items declared in the DASDL with an OCCURS clause must be subscripted.)

Table 1-2 shows the exported names of all data transfer entry points recognized by the DMINTERPRETER Library for each language. The names in this table may be used to declare DMINTERPRETER entry points in application programs, unless renaming is desired or necessary due to the limitations of a particular language.

Table 1-2. Data Transfer Entry Points

COBOL and COBOL74	ALGOL and PL/I**	FORTRAN77
DBGETDISPLAY	ALGOLGETSTRING	FORTRAN77GETCHARACTER
DBGETREAL*	ALGOLGETREAL	FORTRAN77GETREAL
DBGETKANJI	ALGOLGETKANJI	FORTRAN77GETKANJI
DBGETDOUBLE*	ALGOLGETDOUBLE	FORTRAN77GETDOUBLE
DBPUTDISPLAY	ALGOLPUTSTRING	FORTRAN77PUTCHARACTER
DBPUTKANJI	ALGOLPUTKANJI	FORTRAN77PUTKANJI
DBPUTREAL*	ALGOLPUTREAL	FORTRAN77PUTREAL
DBPUTDOUBLE*	ALGOLPUTDOUBLE	FORTRAN77PUTDOUBLE
DBPUTBOOLEAN	ALGOLPUTBOOLEAN	FORTRAN77PUTLOGICAL
DBPUTNULL	ALGOLPUTNULL	FORTRAN77PUTNULL
DBDATA	ALGOLDATA	FORTRAN77DATA

* Available only in COBOL, not in COBOL74.
** PL/I directly calls the ALGOL entry points.

Exception-Handling Entry Points

All DMINTERPRETER entry points except DBEXCEPTIONTEXT and DBEXCEPTIONNAME indicate an exception by returning a particular value. The DMINTERPRETER Library provides three exception handling entry points to aid the programmer in handling and interpreting exceptions:

DBSTATUS

This entry point returns the exception result (category, subcategory, and structure numbers) from the last call on an entry point. Application programs need not directly examine this exception result; instead, they should pass the result to DBEXCEPTIONTEXT or DBEXCEPTIONNAME.

DBEXCEPTIONTEXT

This entry point returns an exception message corresponding to the exception result returned by DBSTATUS. If the variable that holds the exception message is not long enough, then the message is truncated as necessary.

DBEXCEPTIONNAME

This entry point returns the name of the type of exception that occurred.

When an application program calls DBEXCEPTIONTEXT or DBEXCEPTIONNAME, the DMINTERPRETER Library calls the DMSUPPORT Library to retrieve the text or the name of the exception message. DBEXCEPTIONTEXT and DBEXCEPTIONNAME entry points can return any exception that can be returned by the DMSII user language interface. In addition, the DMSUPPORT Library contains a special category of exceptions called INTLIBERRORS, for errors peculiar to the interpretive interface. For more information about exceptions that might occur, refer to the appendix "DMSII Exceptions and Errors."

Table 1-3 shows the exported names of all exception handling entry points recognized by the DMINTERPRETER Library for each language. The names in this table may be used to declare DMINTERPRETER entry points in application programs, unless renaming is desired or necessary due to the limitations of a particular language.

Table 1-3. Exception-Handling Entry Points

COBOL and COBOL74	ALGOL and PL/I *	FORTRAN77
DBSTATUS	ALGOLSTATUS	FORTRAN77STATUS
DBEXCEPTIONTEXT	ALGOLEXCEPTIONTEXT	FORTRAN77EXCEPTIONTEXT
DBEXCEPTIONNAME	ALGOLEXCEPTIONNAME	FORTRAN77EXCEPTIONNAME

* PL/I directly calls the ALGOL entry points.

Attribute-Setting Entry Points

SETLIMIT is the only attribute-setting entry point available. It is used to restrict the number of times the DMINTERPRETER can call the Accessroutines to complete a find. FINDLIMIT is the only valid limit type allowed for the SETLIMIT entry point.

Table 1-4 shows the exported names of the attribute-setting entry point. The names in this table can be used to declare DMINTERPRETER entry points in application programs, unless renaming is desired or necessary due to limitations of a particular language.

Table 1-4. Attribute-Setting Entry Points

COBOL and COBOL74	ALGOL and PL/I *	FORTRAN77
DBSETLIMIT	ALGOLSETLIMIT	FORTRAN77SETLIMIT

* PL/I directly calls the ALGOL entry points.

SAMPLE DATA BASE

The following DASDL input creates EMPJOB, the sample data base that is accessed by all of the sample application programs provided in the appendix "Sample Programs" in this manual:

```
RSTART RESTART DATA SET
(ALPHAID ALPHA(7));
************** GLOBAL RECORD *****************************
FISCAL-YEAR "NEEDS UPDATING EVERY YEAR" NUMBER (6);
POP-EMP POPULATION (50) OF EMP;
DIV-NUMBER NUMBER (5);

******************** EMP *********************************
EMP DATA SET  "MASTER EMPLOYEE FILE"
POPULATION = 50
(
EMP-NO                    NUMBER     (4); %ACTUAL EMP NUMBER
EMP-NAME                  GROUP
    (
    EMP-LNAME                ALPHA      (10);
    EMP-FNAME                ALPHA      (10);
    );
EMP-SSN                   NUMBER      (9);
EMP-DATE-HIRED "YYMMDD"   NUMBER      (6);
EMP-SEC-CLEAR "T=YES"     BOOLEAN        ;
EMP-SALARY "MONTHLY"      NUMBER    (6,2);
            %**** EMP-ED ******
EMP-ED DATA SET "EDUCATIONAL RECORDS FOR EACH EMPLOYEE"
    POPULATION = 10
    (
    ED-DATE "YYMMDD"      NUMBER      (6);
    ED-INST-NAME          ALPHA      (30);
    ED-INST-CODE          NUMBER      (4); %UNIQUE TO SCHOOL
    ED-COURSE             ALPHA      (10);
    ED-GPA                NUMBER    (3,2); %GRADE PT AVR
    );
ED-DATE-SET               SET OF EMP-ED KEY ED-DATE I-S;
EMP-JOB-REF               REFERENCE TO JOB COUNTED;
);
EMP-NO-SET                SET OF EMP KEY EMP-NO I-S;
EMP-LNAME-SET             SET OF EMP
KEY EMP-LNAME DUPLICATES I-S;
EMP-OVER10-SUBSET "VESTED EMPLOYEES" SUBSET OF EMP
                          WHERE (EMP-DATE-HIRED < 700000)
                          BIT VECTOR;
```

```
 ******************* JOB ***************************************
JOB DATA SET "REC TYPE 1 IS CLERICAL, REC TYPE 2 IS NON-CLERICAL"
                        POPULATION = 10
                                (
            JOB-COUNT                COUNT      (10);
            JOB-TYPE                 TYPE        (2);
            JOB-CODE                 NUMBER      (4);
            JOB-TITLE                ALPHA      (30);
         JOB-EMP-SUBSET         SUBSET OF EMP KEY EMP-NO;
                                ),
                                1:
                                (
            TYPE-SPEED               NUMBER      (2);
            SHORTHAND-SPEED          NUMBER      (2);
            CRTUSE-REQD              BOOLEAN        ;
                                ),
                                2:
                                (
            SPANISH-REQD             BOOLEAN        ;
            DRIVING-REQD             BOOLEAN        ;
            JOB-COMMENT              ALPHA      (50);
                                );
       JOB-CODE-SET             SET OF JOB KEY JOB-CODE I-S;
          JOB-MNG-SUBSET "MANAGERS" SUBSET OF JOB WHERE
                                (JOB-CODE > 4999);

 %***************** ADR ***************************************
               ADR DATA SET "MASTER ADDRESS FILE"
                        POPULATION = 50
                                (
            ADR-EMP-NO               NUMBER      (4);
                 ADR-INFO                 GROUP
                                   (
              ADR-STREET-NO             ALPHA      (30);
              ADR-ZIP                   NUMBER      (5);
                                   );
                                );
  ADR-ZIP-SET             SET OF ADR KEY ADR-ZIP I-S DUPLICATES;
```

2 GENERATING THE DMINTERPRETER LIBRARY

The DMINTERPRETER Library can be generated to interface with an entire physical data base, a logical data base, or selected parts of a data base. DMINTERPRETER can be generated for a maximum of 255 structures plus global data.

DMINTERPRETER can be generated from a Work Flow Language (WFL) deck or interactively from a remote terminal.

NOTE

All commands for generating DMINTERPRETER must be entered in uppercase letters.

DMINTERPRETER FILES

DMINTERPRETER is generated using the following files:

1. SYMBOL/BUILDINQ. Symbolic of the program that is compiled to produce SYSTEM/BUILDINQ. SYMBOL/BUILDINQ can be compiled using DMALGOL by equating DMALGOL file TAPE to the file SYMBOL/BUILDINQ.

2. SYSTEM/BUILDINQ. Object code file for SYMBOL/BUILDINQ. SYSTEM/BUILDINQ builds a directory of the portion of the data base that DMINTERPRETER is able to access. This directory, which is named DMIDIRECTORY/<data base name>, is included in the DMINTERPRETER code file for run-time interpretation of the data base structure.

3. DATABASE/DMINTERPRETER. Symbolic file for DMINTERPRETER.

4. SYSTEM/DMINTERFACE. An external co-routine that extracts and checks information from DESCRIPTION/<data base name>.

5. DATABASE/PROPERTIES. Input file to a Data and Structure Definition Language (DASDL) compilation. Each statement in this file defines some aspect of the data base.

6. DESCRIPTION/<data base name>. File containing a complete description of the data base in encoded form. This file is the output from a DASDL compilation.

The generated DMINTERPRETER code file is named DMINTERPRETER/<data base name>. The intermediate compiler input file is named DMIDIRECTORY/<data base name>.

During compilation of the interpretive interface, the format of the DMIDIRECTORY file is checked to ensure that it is consistent with the format of the DMINTERPRETER program symbolic (DATABASE/DMINTERPRETER). If a level mismatch is found between the directory level in the DATABASE/PROPERTIES file and the directory level in the DMIDIRECTORY/<data base name> file, the compiler's error count increases by 1, and the following error message is displayed:

DMIDIRECTORY LEVEL MISMATCH

(The directory level shows the release level. For example, release 33 is recorded as 3300.)

You can correct the problem by rerunning the BUILDINQ program to create a new DMIDIRECTORY file with the correct format.

DMINTERPRETER GENERATION

DMINTERPRETER can be generated from either a remote terminal or a card reader.

The following subsections describe the process of generation.

Generation from Remote Terminals

To begin the generation of the DMINTERPRETER from a remote terminal, enter

```
R BUILDINQ
```

BUILDINQ then responds with the following message:

```
WHAT DATABASE?
```

Continue the generation of the DMINTERPRETER using the following procedure:

1. Enter the data base name in uppercase letters. Alternatively, you can enter $NOZIP and then compile DMINTERPRETER from a job deck. (The $NOZIP option is described in a later part of this section.)

2. If the DESCRIPTION/<data base name> file and the DATABASE/PROPERTIES file are both present, BUILDINQ displays the following message:

```
WHICH SOFTWARE?

1 DMINQUIRY

2 DMINTERPRETER
```

   At this point, enter a 2 to generate the DMINTERPRETER.

3. BUILDINQ then asks:

```
WHICH OPTION?

1 TOTAL DATABASE

2 SELECTED DATASETS

3 LOGICAL DATABASE
```

(If no logical data bases are declared for the data base, option 3 is not displayed.)

Entering 1 enables you to access all data base structures.

Entering 2 enables you to access only selected data base structures. From here, BUILDINQ displays the following message:

```
FOR FOLLOWING DATASETS ENTER:

E (OR EMPTY INPUT) TO EXCLUDE FROM SOFTWARE

I TO INCLUDE INTO SOFTWARE

C TO INDICATE DONE

* TO TERMINATE
```

BUILDINQ then displays the names of all disjoint data sets and awaits a response after each name is displayed. If you enter an asterisk (*), the run is terminated and DMINTERPRETER is not generated. If you enter C, BUILDINQ asks if you want the DBSTRUCTURE included with the data sets. Enter your response. A detailed description of DBSTRUCTURE usage can be found in the section "Determining the Data Base Structure." BUILDINQ then proceeds to step 4.

Entering 3 enables you to access logical data bases. From here, BUILDINQ asks:

```
LOGICAL DATABASE NAME?
```

If you enter a logical data base name with a hyphen in it, BUILDINQ proceeds to step 4 then to step 5. Otherwise, it proceeds to step 4 then to step 6.

4. BUILDINQ then asks:

```
INQUIRY ONLY (YES OR NO)?
```

If your response is YES, DMINTERPRETER is compiled without the ability to create, delete, or update records. If your response is

NO, DMINTERPRETER is compiled with inquiry, create, delete, and update capabilities. If you attempt to open a data base with OPEN ("UPDATE") when update capability has been excluded, an "UNKNOWN OPEN TYPE" error will result. Attempts to create, delete, or update records when update capability has been excluded will result in READONLY errors.

If your data base name has a hyphen in it, BUILDINQ proceeds to step 5 where you rename your data base structure without a hyphen. Otherwise, BUILDINQ proceeds to step 6.

5. At this point, BUILDINQ displays the following message:

   INTERPRETER NAME?

   You must supply a DMINTERPRETER program <user name> that is used to create the file DMINTERPRETER/<user name>. BUILDINQ then proceeds to step 7.

   The <user name> can consist of one to eleven identifiers separated by slashes. Each identifier can contain 1 to 17 alphanumeric characters only. Care must be taken in selecting a <user name> because if there is an existing file with the same name as DMINTERPRETER/<user name>, that existing file is removed.

6. If you chose option 1 in step 3, BUILDINQ displays the following message:

   INTERPRETER NAME (NULL FOR DEFAULT)?

   You can enter a null line to cause BUILDINQ to create a file with the default name DMINTERPRETER/<data base name>, where <data base name> is the name you entered in step 1. Otherwise, you can enter a <user name> as defined in step 5.

7. When the DMINTERPRETER program has been named, BUILDINQ asks the following:

   WHAT QUEUE (NULL FOR DEFAULT)?

   Enter a space to compile DMINTERPRETER in the system default queue. Otherwise, enter a number to compile DMINTERPRETER in the queue corresponding to that number.

8. After the queue is selected, BUILDINQ initiates the compilation of DMINTERPRETER and displays the message:

   # GENERATING DMINTERPRETER

   This is the last step in the DMINTERPRETER generation process.

Remote Generation Using the NOZIP Option

BUILDINQ normally produces a file with the title DMINTERPRETER/<user name> or DMINTERPRETER/<data base name> depending on whether SELECTED DATASETS was chosen in step 5 of the generation process. In this case, BUILDINQ performs a ZIP to compile DMINTERPRETER. However, you can inhibit the ZIP by using the $NOZIP option. You can then compile DMINTERPRETER using a job deck rather than allowing BUILDINQ to ZIP the compilation. The paragraphs.

1. Begin DMINTERPRETER generation as usual. When you see the request:

   WHAT DATABASE?

   Enter the following response:

   $NOZIP

2. BUILDINQ accepts this entry and then repeats the question:

   WHAT DATABASE?

   At this point, enter the data base name. When $NOZIP is used, BUILDINQ does not request the queue specification. If BUILDINQ is run from cards, the QUEUE statement is ignored, if present.

To compile the DMINTERPRETER program after using the $NOZIP option, the following cards must be included in the job deck:

```
COMPILE DMINTERPRETER/<data base name>
     WITH DMALGOL LIBRARY;
COMPILER FILE TAPE = DATABASE/DMINTERPRETER;
COMPILER FILE DASDL = DMIDIRECTORY/<data base name>;
COMPILER FILE PROPERTIES = DATABASE/PROPERTIES;
?  DATA
$ MERGE
?  END
```

The file-equation cards may require modification, depending on the location of the files being used.

Generation from Job Decks

BUILDINQ can also be run using a job deck. The following cards may appear in any order in the job deck:

DATABASE : <data base name>

If this card is present, the data base description file is found in a file titled DESCRIPTION/<data base name>. Otherwise, the BUILDINQ file DASDL must be label-equated to the title of the data base description file.

If this card is present, BUILDINQ produces the DMIDIRECTORY but does not begin compilation of DMINTERPRETER.

DMINTERPRETER

This card must be present to specify the DMINTERPRETER.

LDBNAME : <data base name>

If this card is present, access to the specified logical data base is assumed; otherwise, access to the total data base is assumed. The <data base name> must refer to a logical data base declared in DASDL.

QUEUE : <integer>

If this card is present, DMINTERPRETER is compiled in the specified queue; otherwise, it is compiled in the system default queue.

NOUPDATE

If this card is present, records cannot be updated; otherwise, the ability to update records is included.

The CARD input file for BUILDINQ can be made into a file of type DISK, REMOTE, or READER.

If an error is encountered when the BUILDINQ job deck is run, an appropriate error message is displayed and BUILDINQ terminates.

The following examples show these cards in use in job decks.

Example 1

The following job deck uses the DATABASE card to specify the data base whose description file is used to build DMINTERPRETER:

```
?BEGIN JOB BUILDINQFROMCARDS;
RUN SYSTEM/BUILDINQ;
DATA
DATABASE : MYDB
DMINTERPRETER
QUEUE : 40

?END JOB.
```

Example 2

The following job deck titles the DASDL file for the data base whose description file is used to build DMINTERPRETER:

```
?BEGIN JOB BUILDINQFROMCARDS;
RUN SYSTEM/BUILDINQ;
FILE DASDL (TITLE=DESCRIPTION/MYDB);
DATA
DMINTERPRETER
QUEUE : 40

?END JOB
```

Example 3

The following job deck shows how the interpretive interface can be generated in a batch environment for a data base named DMIDB:

```
?BEGIN JOB CREATE/DMINTERPRETER;
TASK T;
REMOVE DMIDIRECTORY/DMIDB;
RUN SYSTEM/BUILDINQ [T];
DATA
$NOZIP
DATABASE : DMIDB
DMINTERPRETER
? % --- END BUILDINQ INPUT ---
IF T ISNT COMPLETEDOK THEN
ABORT "BUILDINQ RUN FAILED";
COMPILE DMINTERPRETER/DMIDB WITH DMALGOL [T] LIBRARY;
COMPILER FILE DASDL (TITLE=DMIDIRECTORY/DMIDB);
COMPILER FILE CARD  (KIND=PACK,
                     TITLE=DATABASE/DMINTERPRETER);
IF T ISNT COMPILEDOK THEN
ABORT "DMINTERPRETER COMPILE FAILED";
?END JOB.
```

3 ENTRY POINTS IN COBOL/COBOL74

This section explains the COBOL and COBOL74 interfaces to the DMINTERPRETER Library, and provides sample application programs written in these languages to illustrate the following concepts:

1. How variables are declared for the parameters of DMINTERPRETER entry points.

2. How an application program "calls" or "invokes" DMINTERPRETER entry points.

3. How the DMINTERPRETER Library returns results to COBOL and COBOL74 programs.

4. How exceptions returned by entry point calls are handled.

This section also lists all COBOL/COBOL74 entry points of the DMINTERPRETER Library and describes their functions. The following information is included under each description of a COBOL/COBOL74 entry point:

1. A brief explanation of the function of the entry point.

2. Syntax diagrams and semantics that describe required parameters. (More detailed syntax diagrams for data base constructs used with the interpretive interface are given in the appendix "Syntax Diagrams for Entry Point Parameters.")

3. A program fragment that illustrates the declaration of variables for the parameters of the entry point.

4. A program fragment that illustrates the invocation of the entry point.

All explanations, requirements, diagrams, and program fragments apply to both COBOL and COBOL74 unless a difference between the two languages is noted. Table 3-1 summarizes the COBOL/COBOL74 entry points, and lists them in the order in which they are described later in this section.

Table 3-1. Summary of
COBOL/COBOL74 Entry Points

Standard Entry Points:

DBOPEN
DBCLOSE
DBBEGINTRANSACTION
DBENDTRANSACTION
DBCREATE
DBRECREATE
DBSTORE
DBFREE
DBSET
DBFIND
DBLOCK
DBDELETE
DBVERB

Data Transfer Entry Points:

DBGETDISPLAY
DBGETKANJI
DBGETREAL
DBGETBOOLEAN
DBGETDOUBLE
DBPUTDISPLAY
DBPUTKANJI
DBPUTREAL
DBPUTBOOLEAN
DBPUTDOUBLE
DBPUTNULL
DBDATA

Exception-Handling Entry Points:

DBSTATUS
DBEXCEPTIONTEXT
DBEXCEPTIONNAME

Attribute-Setting Entry Point:

DBSETLIMIT

THE COBOL/COBOL74 INTERFACE TO DMINTERPRETER

The first step in writing an application program that accesses the interpretive interface is to provide for the linkage of entry points called by an application program to entry points of the DMINTERPRETER Library. In COBOL and COBOL74, this is accomplished by changing the TITLE attribute of the library to the actual name of the library code file.

The following syntax explains how to change the TITLE attribute for libraries within a COBOL or COBOL74 program.

Syntax

```
-- CHANGE -- ATTRIBUTE -- TITLE --- OF --- " --<library- ID>-- " -->
                                    |      |
                                    |- IN -|

>- TO ---<identifier>-----------------------------------------------|
       |                        |
       |-<non-numeric literal>-|
```

Explanation

The library ID is the file name of the library to be called, which should be composed of letters, numbers, hyphens, underscores, and slashes. If the library ID does not contain slashes, the entire name can be used as the internal name (INTNAME) for the library; if the library ID does contain slashes, then the identifier that follows the last slash is used as the INTNAME for the library.

The library ID including slashes is used as the TITLE of the library. The CHANGE construct changes the library ID to an identifier or a non-numeric literal within the COBOL or COBOL74 application program, so that every time the program makes a call on the library ID, the call is directed to the library whose name corresponds to the identifier or the non-numeric literal.

In the sample COBOL and COBOL74 programs provided in the appendix "Sample Programs," linkage to the DMINTERPRETER Library has been accomplished in the following manner:

Declaration of Identifier:

```
01 DMI-NAME PIC X(24) VALUE IS "DMINTERPRETER/EMPJOB".
```

Linking to DMINTERPRETER Library:

```
CHANGE ATTRIBUTE TITLE OF "DMINTERPRETER" TO DMI-NAME.
```

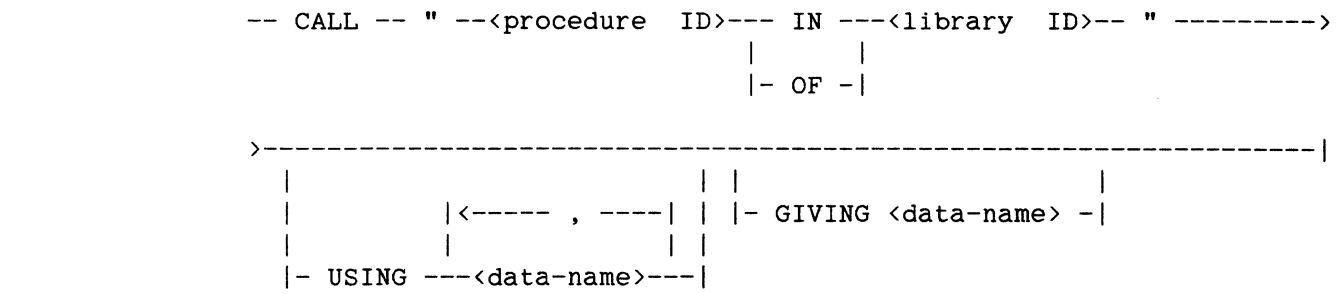
Although the DATABASE/DMINTERPRETER symbolic file contains declarations of library entry points for other interface languages, the COBOL and COBOL74 languages do not require explicit declaration of library entry points. The exported names of COBOL/COBOL74 entry points (given in DATABASE/DMINTERPRETER) with the "DB" prefix must be used in COBOL and COBOL74 application programs, because there are no renaming facilities in COBOL and COBOL74 as there are in other interface languages. The parameters for COBOL/COBOL74 entry points are translated into ALGOL-compatible parameters, and the ALGOL entry points within the DMINTERPRETER Library perform the actual operations against the data base.

INVOCATION OF ENTRY POINTS

In COBOL and COBOL74, entry points of the DMINTERPRETER Library are invoked using a call non-numeric literal statement, which passes the required parameters and receives a result. The following syntax describes the COBOL/COBOL74 CALL statement, where procedure ID is the name of the procedural entry point, and library ID is the name of the library to be called.

Syntax

<call non-numeric literal statement>



Explanation

Parameters for the COBOL/COBOL74 entry point are supplied in the USING clause of the CALL statement, and the result in the GIVING clause. Parameters for the standard entry points, and for most of the data transfer entry points, must be DISPLAY items declared at level 01. However, when using COBOL, the following data transfer entry points must be declared as COMP-4 or COMP-5 items at level 77:

DBGETREAL	DBPUTREAL
DBGETDOUBLE	DBPUTDOUBLE

When using COBOL74, these data transfer points must be declared as REAL or DOUBLE items at level 77. Detailed explanations of how to use the required parameters for each entry point are given under the individual entry point descriptions in this section.

After a call on an entry point of the DMINTERPRETER Library, a result descriptor is returned as an integer into the result variable that the user program supplies in the GIVING clause of the CALL statement. The integer returned by an entry point will always be a 0 to indicate success, or a 1 to indicate that an exception has occurred.

For COBOL, the result variable must be a COMP-2 item declared at level 01. For COBOL74, the result variable must be a COMP item declared at level 01. If an exception has occurred, the program must then call the exception handling entry point named DBSTATUS, along with DBEXCEPTIONNAME or DBEXCEPTIONTEXT to find out the name of the exception type or the text of the exception message respectively. For more information on exceptions, refer to "Exception-Handling Entry Points" later in this section.

3.1 STANDARD ENTRY POINTS

Standard entry points correspond to functions performed by Data Management System II (DMSII) user language interface statements. The results returned by standard entry points are used for exception handling.

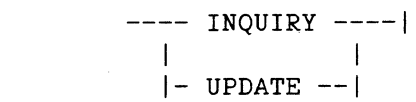
DBOPEN

DBOPEN opens the data base for inquiry or update, depending on which parameter is used. DBOPEN must be called before any data base operations are performed. DBOPEN, the exception handling entry points (DBSTATUS, DBEXCEPTIONTEXT, and DBEXCEPTIONNAME), and operations on the intrinsic data set DBSTRUCTURE are the only functions allowed when the data base is closed.

Required Parameters:	Data Type of Parameters:
<open disposition>	DISPLAY

Syntax

<open disposition>



Explanation

INQUIRY

Causes the data base to be opened for INQUIRY. No updates are allowed.

UPDATE

Causes the data base to be opened for UPDATE. Both inquiry and update are allowed.

Example

The following program fragments show how DBOPEN is used to open the data base for UPDATE. The OPEN-TYPE variable must be set to UPDATE or INQUIRY before issuing the CALL statement.

Declaration of Variables:

```
01  OPEN-TYPE    PIC X(6) VALUE "UPDATE".
```

Invocation of Entry Point:

```
CALL "DBOPEN OF DMINTERPRETER"
     USING OPEN-TYPE
     GIVING RESULT.
```

DBCLOSE

DBCLOSE explicitly closes the data base. (The data base is implicitly closed when the library execution completes.)

Required Parameters:	Data Type of Parameters:
None	Not Applicable

Example

The following example shows how DBCLOSE is used to close the data base. When calling DBCLOSE, a USING clause need not be specified, because there are no required parameters for DBCLOSE.

Declaration of Variables:

Not applicable when using DBCLOSE.

Invocation of Entry Point:

```
CALL "DBCLOSE OF DMINTERPRETER"
     GIVING RESULT.
```

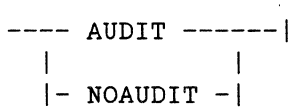
DBBEGINTRANSACTION

DBBEGINTRANSACTION causes the user program to enter transaction state. The audit parameter specifies whether an image of the current restart data set record is to be audited. Calls on DBBEGINTRANSACTION are ignored for unaudited data bases.

Required Parameters:	Data Type of Parameters:
<audit>	DISPLAY

Syntax

<audit>



Explanation

AUDIT

Specifies that an image of the current restart data set will be audited.

NOAUDIT

Specifies that auditing an image of the current restart data set is not required. NOAUDIT is the default value of the audit parameter.

Example

The following example shows how a user program can enter transaction state by calling DBBEGINTRANSACTION. The audit parameter must be set to AUDIT or NOAUDIT before issuing the CALL statement.

Declaration of Variables:

```
01  P-AUDIT          PIC X(5) VALUE IS "AUDIT".
```

Invocation of Entry Point:

```
CALL "DBBEGINTRANSACTION OF DMINTERPRETER"
      USING P-AUDIT
      GIVING RESULT.
```

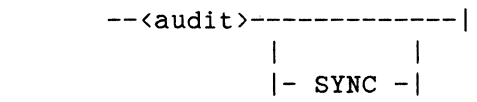
DBENDTRANSACTION

DBENDTRANSACTION causes the user program to leave transaction state. DBENDTRANSACTION should be used only with audited data bases. Calls on DBENDTRANSACTION are ignored for unaudited data bases.

Required Parameters:	Data Types of Parameters:
<auditsync>	DISPLAY

Syntax

<auditsync>



Explanation

<audit>

Specifies whether an image of the current restart data set will be audited (this is the audit parameter discussed under DBBEGINTRANSACTION in this section).

<audit> SYNC

Forces a syncpoint, when an image of the current restart data set is audited.

Example

The following program fragments show how DBENDTRANSACTION is used to cause a program to exit transaction state, with the current restart data set record being audited. The P-AUDIT variable must be set to AUDIT or AUDIT SYNC before issuing the CALL statement.

Declaration of Variables:

```
01  P-AUDIT     PIC X(5) VALUE IS "AUDIT".
```

Invocation of Entry Point:

```
CALL "DBENDTRANSACTION OF DMINTERPRETER"
     USING P-AUDIT
     GIVING RESULT.
```

DBCREATE

DBCREATE creates a new record for the specified data set. The record type parameter specifies whether the record will be fixed-length or variable-format.

Required Parameters:	Data Types of Parameters:
<data set>	DISPLAY
<record type>	DISPLAY

Syntax

<data set>

```
--<qualified name>--|
```

<record type>

```
-- " ---<integer>--- " --|
      |           |
      |-<blank>---|
```

Explanation

<data set>

Names a data set for which a new record will be created.

"<integer>"

A non-zero integer specifies that the record will be variable-length. If the specified integer is 0, the fixed part of a variable-length record is created.

"<blank>"

If an empty string is specified, a fixed-length record will be created.

Example

The following program fragments show how DBCREATE is used to create a new fixed-length record belonging to a data set named EMP.

Declaration of Variables:

```
01  DATA-SET-NAME          PIC X(17) VALUE IS "EMP".
01  SPACE-1                PIC X(1) VALUE IS " ".
```

Invocation of Entry Point:

```
CALL "DBCREATE OF DMINTERPRETER"
      USING DATA-SET-NAME, SPACE-1
      GIVING RESULT.
```

DBRECREATE

DBRECREATE re-creates, or copies, a new image of an existing record for the purpose of modification. The data set parameter names the existing data set where the record resides. The record type parameter specifies whether the re-created record will be fixed-length or variable-format.

Required Parameters:	Data Types of Parameters:
<data set>	DISPLAY
<record type>	DISPLAY

Syntax

<data set>

```
--<qualified name>--|
```

<record type>

```
-- " ---<integer>--- " --|
      |           |
      |-<blank>---|
```

Explanation

<data set>

Names a data set for which a record will be re-created.

"<integer>"

A non-zero integer specifies that the record will be variable-length. If the specified integer is 0, the fixed part of a variable-length record is created.

"<blank>"

If an empty string is specified, a fixed-length record will be created.

Example

The following program fragments show how DBRECREATE is used to re-create an image of an existing fixed-length record belonging to a data set named EMP.

Declaration of Variables:

```
01  DATA-SET-NAME          PIC X(17) VALUE IS "EMP".
01  SPACE-1                PIC X(1) VALUE IS " ".
```

Invocation of Entry Point:

```
CALL "DBRECREATE OF DMINTERPRETER"
      USING DATA-SET-NAME, SPACE-1
      GIVING RESULT.
```

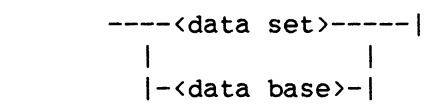
DBSTORE

DBSTORE stores a record in the specified structure. For audited data bases, DBBEGINTRANSACTION must have been called before DBSTORE is called, because the program must be in transaction state in order for a record to be stored. If DBSTORE is called outside of transaction state for an audited data base, an exception is returned.

Required Parameters:	Data Types of Parameters:
<structure>	DISPLAY

Syntax

<structure>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

Example

The following program fragments show how DBSTORE is used to store a record in a data set named EMP.

Declaration of Variables:

```
01  DATA-SET-NAME        PIC X(17) VALUE IS "EMP".
```

Invocation of Entry Point:

```
CALL "DBSTORE OF DMINTERPRETER"
      USING DATA-SET-NAME
      GIVING RESULT.
```

DBFREE

DBFREE unlocks a data set record or a global data record. DBFREE may be used to unlock a record that you anticipate may not be implicitly freed for a long time. After DBFREE is called for a record, other programs may lock that record.

Required Parameters:	Data Types of Parameters:
<structure>	DISPLAY

Syntax

<structure>

```
----<data set>-----|
  |               |
  |-<data base>-|
```

Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

Example

The following program fragments show how DBFREE is used to unlock a record belonging to a specified structure. In this case, a record in the data set EMP is being unlocked.

Declaration of Variables:

```
01  DATA-SET-NAME        PIC X(17) VALUE IS "EMP".
```

Invocation of Entry Point:

```
CALL "DBFREE OF DMINTERPRETER"
      USING DATA-SET-NAME
      GIVING RESULT.
```

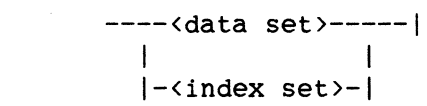
DBSET

DBSET sets particular data sets or index sets to their beginning or ending points, or sets the position pointer of an index set to the current record of a particular data set. The following syntax diagrams and semantics explain which structures can be set to which targets.

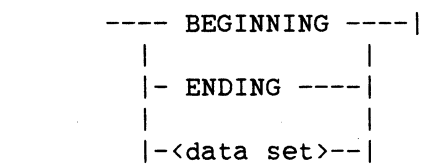
Required Parameters:	Data Type of Parameters:
<structure>	DISPLAY
<target>	DISPLAY

Syntax

<structure>



<target>



Explanation

<data set> BEGINNING

Names a particular data set that will be set to its beginning point at run time.

<data set> ENDING

Names a particular data set that will be set to its ending point at run time.

<index set> <data set>

Names the data set spanned by the index set, and sets the position pointer of the index set to the current record within the spanned data set.

<index set> BEGINNING

Names a particular index set that will be set to its beginning point at run time.

<index set> ENDING

Names a particular index set that will be set to its ending point at run time.

Example

The following program fragments show how DBSET is used to set a particular data base structure to a desired target. In this case, the data set named EMP is set to its beginning point.

Declaration of Variables:

```
01  DATA-SET-NAME        PIC X(17) VALUE IS "EMP".
01  BEGIN-1              PIC X(9) VALUE IS "BEGINNING".
```

Invocation of Entry Point:

```
CALL "DBSET OF DMINTERPRETER"
      USING DATA-SET-NAME, BEGIN-1
      GIVING RESULT.
```

DBFIND

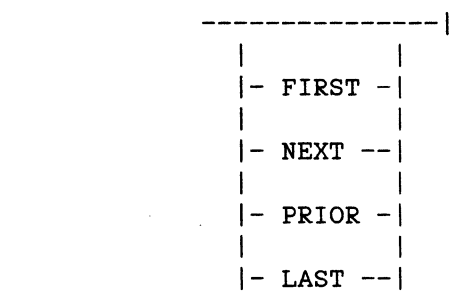
DBFIND transfers a record to the DMSII user work area associated with the specified data base structure. The other parameters shown below indicate the direction in which the FIND operation proceeds, and the selection expression used during the search.

A DBFIND call is more sophisticated than a user language interface FIND statement. Several restrictions have been eased, and some algorithmic enhancements have been made. These improvements are discussed in detail in the appendix "DBFIND Enhancements."

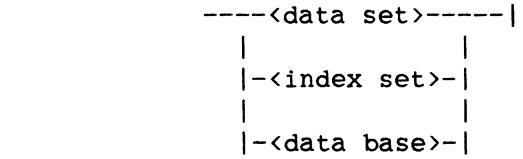
Required Parameters:	Data Type of Parameters:
<direction>	DISPLAY
<structure>	DISPLAY
<condition>	DISPLAY

Syntax

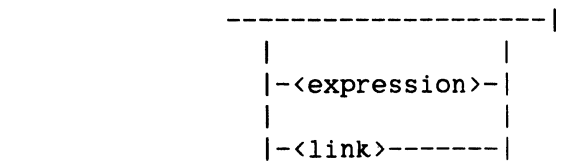
<direction>



<structure>



<condition>



Explanation

CURRENT

Indicates that the FIND operation will transfer the current record in the specified structure to the user work area.

FIRST

Indicates that the FIND operation will transfer the first record in the specified structure to the user work area.

NEXT

Indicates that the FIND operation will transfer the next record in the specified structure to the user work area.

PRIOR

Indicates that the FIND operation will transfer the prior record in the specified structure to the user work area.

LAST

Indicates that the FIND operation will transfer the last record in the specified structure to the user work area.

<data set>

Names a data set in the data base that will be accessed at run time.

<index set>

Names an index set in the data base that will be used for the search.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<expression>

A selection expression used during the DBFIND operation to find records that meet a specified condition.

<link>

Specifies the data base structure or data set item that links the current record to a record in another data set.

Example

The following program fragments show how DBFIND is used to find the first record in a data set named EMP, which satisfies a specified condition.

Declaration of Variables:

```
01  FIRST-1              PIC X(5) VALUE "FIRST".
01  DATA-SET-NAME        PIC X(17) VALUE IS "EMP".
01  CONDITION-1          PIC X(9) VALUE IS "EMP-NO=11".
```

Invocation of Entry Point:

```
CALL "DBFIND OF DMINTERPRETER"
      USING FIRST-1, DATA-SET-NAME, CONDITION-1
      GIVING RESULT.
```

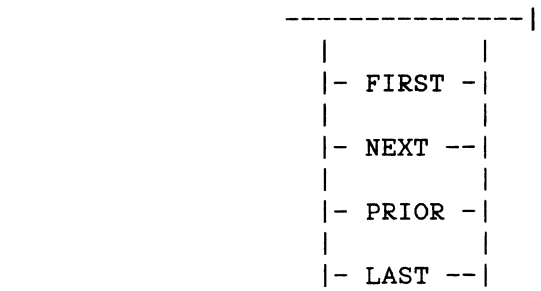
DBLOCK

DBLOCK functions in the same manner as DBFIND, except that a record is locked after it is found. DBLOCK transfers a record to the user work area associated with the specified data structure and locks it there for modification. The other parameters shown below indicate the direction in which the search for the record proceeds, and the selection expression used during the search.

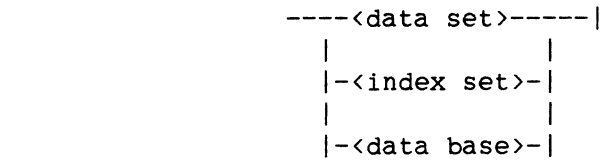
Required Parameters:	Data Type of Parameters:
<direction>	DISPLAY
<structure>	DISPLAY
<condition>	DISPLAY

Syntax

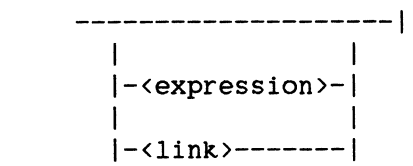
<direction>



<structure>



<condition>



Explanation

(empty)

If the direction parameter is empty, the default value is used, and the DBLOCK will transfer the current record in the specified structure to the user work area.

FIRST

Indicates that the DBLOCK operation will transfer the first record in the specified structure to the user work area.

NEXT

Indicates that the DBLOCK operation will transfer the next record in the specified structure to the user work area.

PRIOR

Indicates that the DBLOCK operation will transfer the prior record in the specified structure to the user work area.

LAST

Indicates that the DBLOCK operation will transfer the last record in the specified structure to the user work area.

<data set>

Names a data set in the data base that will be accessed at run time.

<index set>

Names an index set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<expression>

A selection expression used during the DBLOCK operation to find and lock records that meet a specified condition.

<link>

Specifies the data base structure or data set item that links the current record to a record in another data set.

Example

The following program fragments show how DBLOCK is used to find and lock the first record in a data set named EMP, which satisfies a specified condition.

Declaration of Variables:

```
01  FIRST-1              PIC X(5) VALUE "FIRST".
01  DATA-SET-NAME        PIC X(17) VALUE IS "EMP".
01  CONDITION-1          PIC X(9) VALUE IS "EMP-NO=11".
```

Invocation of Entry Point:

```
CALL "DBLOCK OF DMINTERPRETER"
      USING FIRST-1, DATA-SET-NAME, CONDITION-1
      GIVING RESULT.
```

DBDELETE

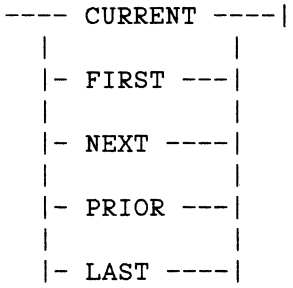
DBDELETE deletes a record from the specified data set. The parameters shown below indicate the direction in which the search and deletion operation proceeds, and the selection expression used during the search.

DBDELETE functions in the same manner as DBFIND, except that a record is deleted after it is found, unless it is a global data record which cannot be deleted. Whenever DBDELETE is called, a locking function is implicitly performed to lock the data set record before it is deleted.

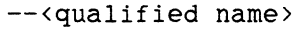
Required Parameters:	Data Type of Parameters:
<direction>	DISPLAY
<data set>	DISPLAY
<condition>	DISPLAY

Syntax

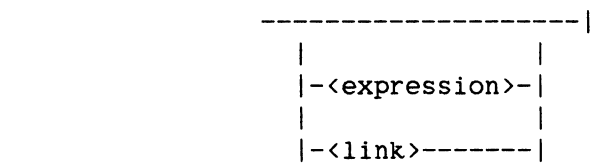
<direction>



<data set>



<condition>



Explanation

CURRENT

Indicates that the DBDELETE call will transfer the current record in the specified structure to the user work area before the record is deleted.

FIRST

Indicates that the DBDELETE call will transfer the first record in the specified structure to the user work area before the record is deleted.

NEXT

Indicates that the DBDELETE call will transfer the next record in the specified structure to the user work area before the record is deleted.

PRIOR

Indicates that the DBDELETE call will transfer the prior record in the specified structure to the user work area before the record is deleted.

LAST

Indicates that the DBDELETE call will transfer the last record in the specified structure to the user work area before the record is deleted.

<qualified name>

See the appendix "Syntax Diagrams for Entry Point Parameters" for basic data base constructs that explain the meaning of qualified name in detail.

<expression>

A selection expression used during the DBDELETE operation to find and delete records that meet a specified condition.

<link>

Specifies the data base structure or data set item that links the current record to a record in another data set.

Example

The following program fragments show how DBDELETE is used to find and delete the first record in a data set named EMP, which satisfies a specified condition.

Declaration of Variables:

```
01  FIRST-1              PIC X(5) VALUE "FIRST".
01  DATA-SET-NAME        PIC X(17) VALUE IS "EMP".
01  CONDITION-1          PIC X(9) VALUE IS "EMP-NO=11".
```

Invocation of Entry Point:

```
CALL "DBDELETE OF DMINTERPRETER"
      USING FIRST-1, DATA-SET-NAME, CONDITION-1
      GIVING RESULT.
```

DBVERB

DBVERB causes a specified user language interface statement to be performed. The parameters that are used must evaluate to a complete statement. Permissible DMSII user language constructs are shown in the syntax diagrams below, but exception actions and input/output mapping clauses are not allowed. Brief descriptions of these statements are given below, but full descriptions are given in the DMSII User Language Interface Programming Guide.

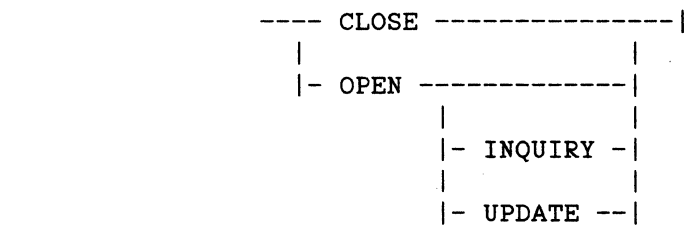
DBVERB allows flexible, run-time definition of DMSII operations, and provides users with an easy transition from DMSII user language interface statements to DMINTERPRETER statements. But because it must completely parse its input, DBVERB runs at higher processor cost than the other, more specialized entry points.

Required Parameters:
-------- ----------

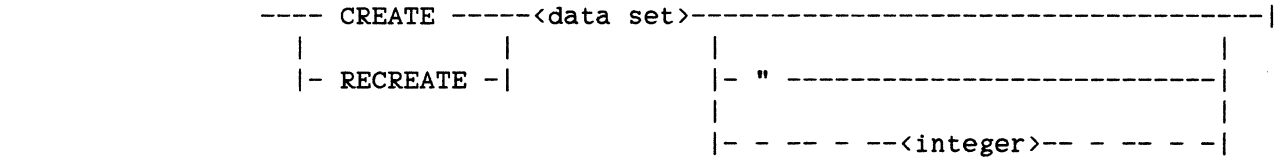
<user language interface statement>

Syntax

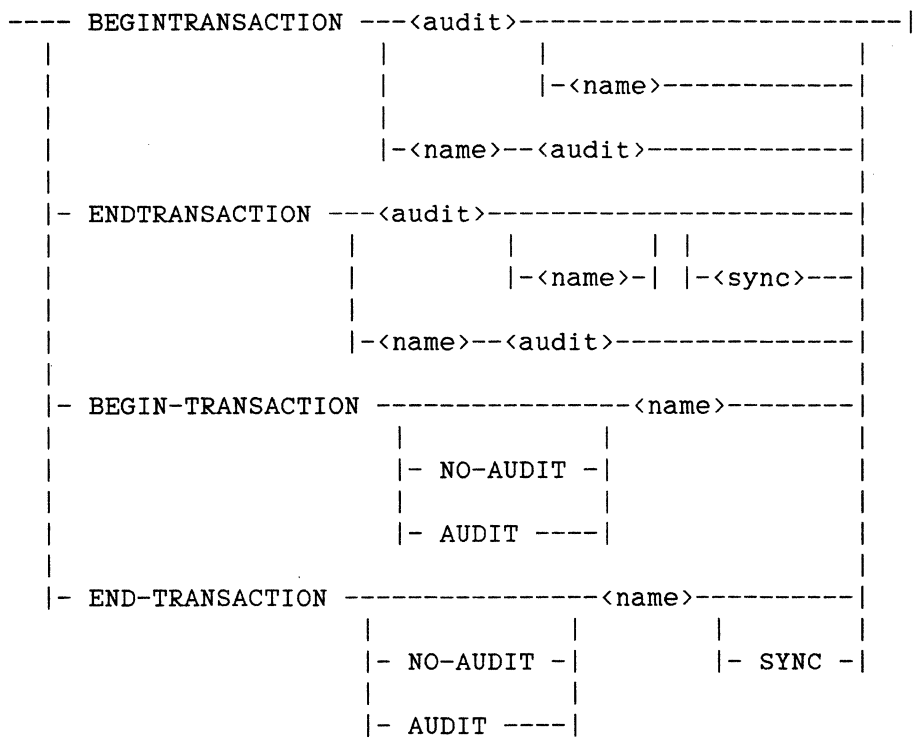
<data base statement>



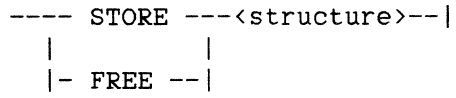
<create statement>



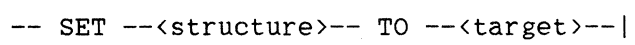
<transaction statement>



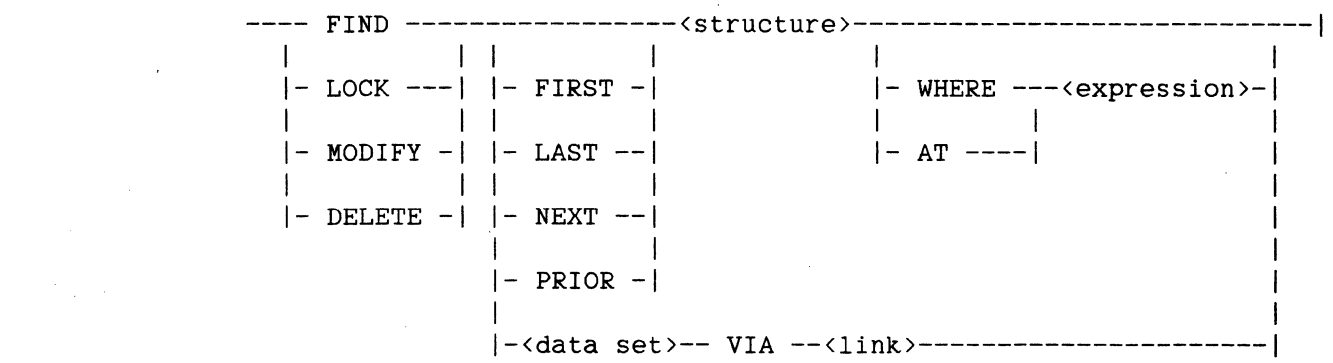
<record statement>



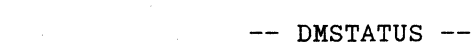
<set statement>



<search statement>



<status statement>



Explanation

<data base statement>

Closes the data base, or opens the data base for inquiry or update.

<create statement>

Initializes the user work area for an existing data set record or a re-created image of a data set record.

<transaction statement>

Causes a program to enter transaction state or exit transaction state, with or without auditing an image of the current restart data set and forcing a syncpoint. The name specifies the restart data set, which is checked for validity when it is present.

<record statement>

Stores a new or modified record in a specified structure, or unlocks a record residing in a specified structure.

<set statement>

Sets a specified structure to its beginning or ending, or to a particular data set if the structure is an index set.

<search statement>

Finds a selected record in a specified structure, and locks the record for modification or deletion.

NOTE

The DMSTATUS form of a call on DBVERB can be used in other languages, but not in COBOL and COBOL74, because these languages cannot pass an exception parameter. Instead, COBOL and COBOL74 programs should directly call the DBSTATUS entry point.

Example

The following program fragments show how DBVERB can be called to execute a user language interface statement.

Declaration of Variables:

```
01  DATA-REQUEST       PIC X(40) VALUE IS SPACES.
```

Invocation of Entry Point:

```
*         USING DBVERB ON A SET STATEMENT
*
       MOVE "SET EMP TO BEGINNING" TO DATA-REQUEST.
       CALL "DBVERB OF DMINTERPRETER"
             USING DATA-REQUEST
             GIVING RESULT.
       IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
*
*          USING DBVERB ON A SEARCH STATEMENT
*
       MOVE "FIND FIRST EMP AT EMP-NO=11" TO DATA-REQUEST.
       CALL "DBVERB OF DMINTERPRETER"
             USING DATA-REQUEST
             GIVING RESULT.
       IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
```

3.2 DATA TRANSFER ENTRY POINTS

Data transfer entry points, prefixed with "GET" or "PUT," are provided for moving data to and from user work areas associated with specified data base structures. The "GET" entry points move data from the data base to variables that have been declared by the programmer. The "PUT" entry points move data from variables declared by the programmer to the data base. Each item needed from a data set record must be individually moved to a COBOL variable, by calling the DMINTERPRETER Library, before the item can be manipulated by the program.

The DMINTERPRETER Library checks the parameters of data transfer entry points to ensure that the requested operations are compatible with the DASDL data base description. For example, items declared in the data base description with an OCCURS clause must be subscripted.

The "PUT" entry points (DBPUTDISPLAY, DBPUTREAL, DBPUTBOOLEAN, DBPUTDOUBLE, and DBPUTNULL) return an exception if the value to be put into the data base does not fit into the specified data item. The "GET" entry points (DBGETDISPLAY, DBGETREAL, DBGETDOUBLE, and DBGETBOOLEAN) return an exception if the declared variable is too short to contain the specified data item.

DBGETDISPLAY

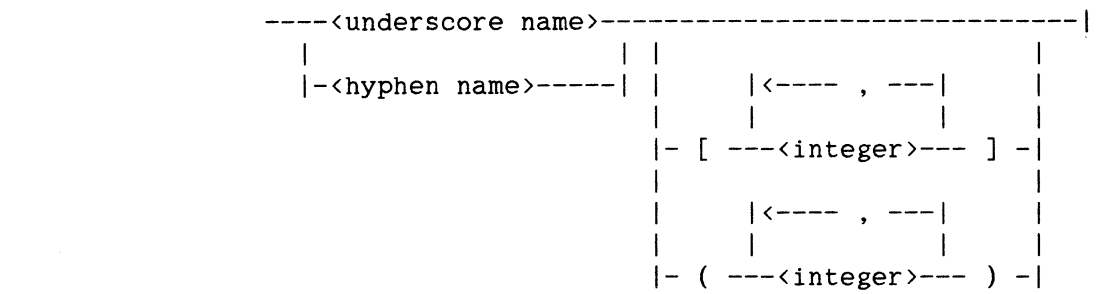
DBGETDISPLAY sets a variable declared in the application program to a character string representing the contents of an ALPHA item or GROUP item from the current record in the specified structure.

Required Parameters:	Data Type of Parameters:
<structure>	DISPLAY
<item name>	DISPLAY
<variable>	DISPLAY

Syntax

<structure>

<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

A COBOL/COBOL74 variable that holds the character string from the data base.

Example

The following program fragments show how DBGETDISPLAY is used to move a data item value from a DMSII user work area to a variable declared by the programmer.

Declaration of Variables:

```
01  DATA-SET-NAME        PIC X(17) VALUE IS "EMP".
01  ITEM-NAME            PIC X(17) VALUE IS SPACES.
01  VAR-1                PIC X(10) VALUE IS SPACES.
```

Invocation of Entry Point:

```
MOVE "EMP-FNAME" TO ITEM-NAME.
CALL "DBGETDISPLAY OF DMINTERPRETER"
    USING DATA-SET-NAME,ITEM-NAME,VAR-1
    GIVING RESULT.
```

DBGETKANJI

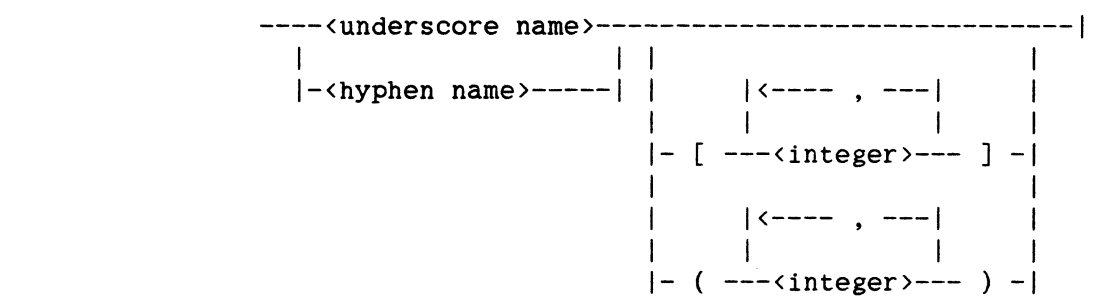
DBGETKANJI sets a variable declared in the application program to a Kanji character string representing the contents of an ALPHA item of which USAGE IS KANJI from the current record in the specified structure.

Required Parameters:	Data Type of Parameters:
<structure>	DISPLAY
<item name>	DISPLAY
<variable>	DISPLAY or KANJI

Syntax

<structure>

<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

A COBOL/COBOL74 variable that holds the character string from the data base and may be defined as PIC X(2n) or PIC X(n) USAGE IS KANJI.

Example

The following program fragments show how DBGETKANJI is used to move a data item value from a DMSII user work area to a variable declared by the programmer.

Declaration of Variables:

```
01  DATA-SET-NAME        PIC X(17) VALUE IS "EMP".
01  ITEM-NAME            PIC X(17) VALUE IS SPACES.
01  VAR-1                PIC X(10) USAGE IS KANJI,
                                   VALUE IS SPACES.
```

Invocation of Entry Points:

```
MOVE "EMP-KNAME" TO ITEM-NAME.
CALL "DBGETKANJI OF DMINTERPRETER"
    USING DATA-SET-NAME,ITEM-NAME,VAR-1
    GIVING RESULT.
```

DBGETREAL

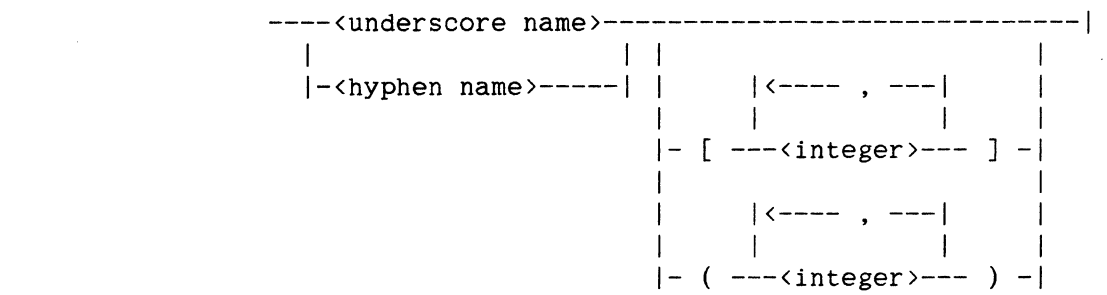
DBGETREAL sets a variable declared in the program to the numeric value of a NUMBER, REAL, FIELD, COUNT, RECORDTYPE, or POPULATION data item in the current record of the specified data base structure. For COBOL, the parameter for the real value must be declared in the working storage section of the program as a level 77 COMP-4 item. For COBOL74, the parameter for the real value must be declared in the working storage section of the program as a level 77 REAL item. If an exception occurs, the variable is set to HIGH-VALUE (all bits ON), and an appropriate exception is returned. Numeric items that are too large to be retrieved by DBGETREAL can be retrieved using the DBGETDOUBLE entry point.

Required Parameters:	Data Type of Parameters:
<structure>	DISPLAY
<item name>	DISPLAY
<variable>	COMP-4 (for COBOL) or REAL (for COBOL74)

Syntax

<structure>

<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

A COBOL real variable.

Example

The following program fragments show how DBGETREAL is used to move a data item value from a DMSII user work area to a variable declared by the programmer.

Declaration of Variables:

```
01  DATA-SET-NAME        PIC X(17) VALUE IS "EMP".
01  ITEM-NAME            PIC X(17) VALUE IS SPACES.
77  VAR-2                COMP-4.
```

Invocation of Entry Point:

```
MOVE "EMP-NO" TO ITEM-NAME.
CALL "DBGETREAL OF DMINTERPRETER"
    USING DATA-SET-NAME,ITEM-NAME,VAR-2
    GIVING RESULT.
```

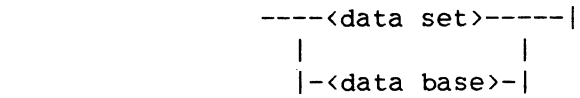
DBGETBOOLEAN

DBGETBOOLEAN retrieves a Boolean value from a Boolean item or a bit in a FIELD item, and sets a COBOL variable to either the string "TRUE" or the string "FALSE" depending on the value. The COBOL variable will be set to a false indication if an exception occurs. The character string that is returned will be truncated if the declared variable is too short.

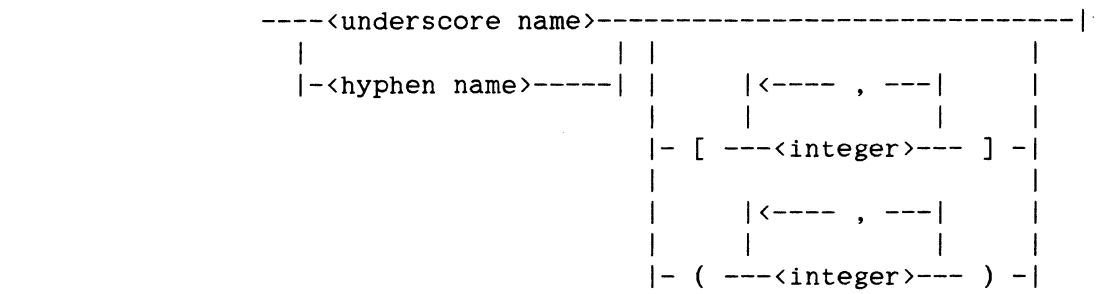
Required Parameters:	Data Type of Parameters:
<structure>	DISPLAY
<item name>	DISPLAY
<variable>	DISPLAY

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

A COBOL display variable that holds the string "TRUE" or the string "FALSE".

Example

The following program fragments show how DBGETBOOLEAN is used to move a data item value from the user work area to a variable declared by the programmer.

Declaration of Variables:

```
01  DATA-SET-NAME        PIC X(17) VALUE IS "EMP".
01  ITEM-NAME            PIC X(17) VALUE IS SPACES.
01  VAR-1                PIC X(10) VALUE IS SPACES.
```

Invocation of Entry Point:

```
MOVE "EMP-SEC-CLEAR" TO ITEM-NAME.
CALL "DBGETBOOLEAN OF DMINTERPRETER"
     USING DATA-SET-NAME,ITEM-NAME,VAR-1
     GIVING RESULT.
```

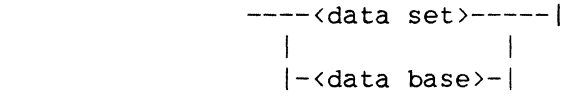
DBGETDOUBLE

DBGETDOUBLE retrieves numeric data items whose number of significant digits exceeds the capacity of a real variable. DBGETDOUBLE sets a variable declared in the program to the numeric value of a NUMBER, REAL, FIELD, COUNT, RECORDTYPE, or POPULATION data item in the current record of the specified structure. For COBOL, the parameter for the numeric value must be declared in the working storage section of the program as a level 77 COMP-5 item. For COBOL74, the parameter for the numeric value must be declared in the working storage section of the program as a level 77 DOUBLE item. If an exception occurs, the declared variable is set to HIGH-VALUE (all BITS on), and an appropriate exception is returned.

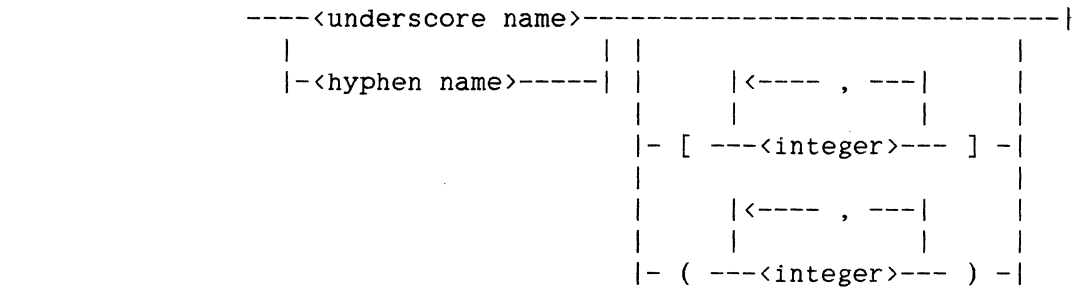
Required Parameters:	Data Type of Parameters:
<structure>	DISPLAY
<item name>	DISPLAY
<variable>	COMP-5 (for COBOL)
	or
	DOUBLE (for COBOL74)

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

A COBOL variable that can hold a double-precision numeric value.

Example

The following program fragments show how DBGETDOUBLE is used to move a data item value from a DMSII user work area to a variable declared by the programmer.

Declaration of Variables:

```
01  DATA-SET-NAME        PIC X(17) VALUE IS "EMP".
01  ITEM-NAME            PIC X(17) VALUE IS SPACES.
77  VAR-3                COMP-5.
```

Invocation of Entry Point:

```
MOVE "EMP-SSN" TO ITEM-NAME.
CALL "DBGETDOUBLE  OF DMINTERPRETER"
    USING DATA-SET-NAME,ITEM-NAME,VAR-3
    GIVING RESULT.
```

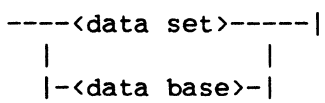
DBPUTDISPLAY

DBPUTDISPLAY puts a string into a particular data item in the current record of the specified structure. The data item must be declared in DASDL as an ALPHA or GROUP item. The string must not exceed the length of the data item where it will be stored, otherwise an exception is returned.

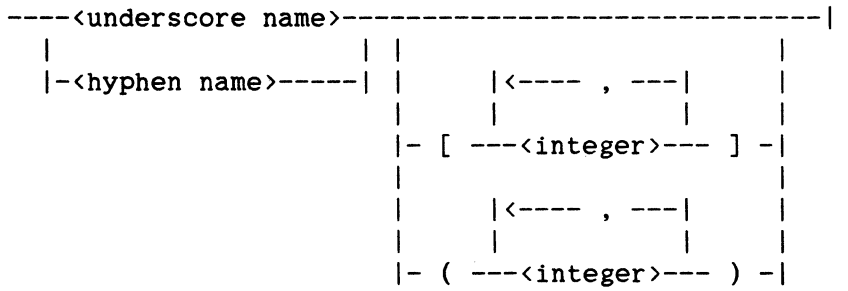
Required Parameters:	Data Type of Parameters:
<structure>	DISPLAY
<item name>	DISPLAY
<variable>	DISPLAY

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

<variable>

A COBOL or COBOL74 variable that holds the string to be put into the data base.

Example

The following program fragments show how DBPUTDISPLAY is used to move a data item value from a string variable declared by the programmer to a DMSII user work area.

Declaration of Variables:

```
01  DATA-SET-NAME        PIC X(17) VALUE IS "EMP".
01  ITEM-NAME            PIC X(17) VALUE IS SPACES.
01  VAR-1                PIC X(10) VALUE IS SPACES.
```

Invocation of Entry Point:

```
MOVE "EMP-FNAME" TO ITEM-NAME.
MOVE "CATHY" TO VAR-1.
CALL "DBPUTDISPLAY OF DMINTERPRETER"
    USING DATA-SET-NAME,ITEM-NAME,VAR-1
    GIVING RESULT.
```

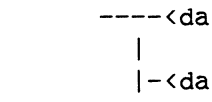
DBPUTKANJI

DBPUTKANJI puts a string into a particular data item in the current record of the specified structure. The data item must be declared in DASDL as an ALPHA item of which USAGE IS KANJI. The string must not exceed the length of the data item where it will be stored; otherwise, an exception is returned.

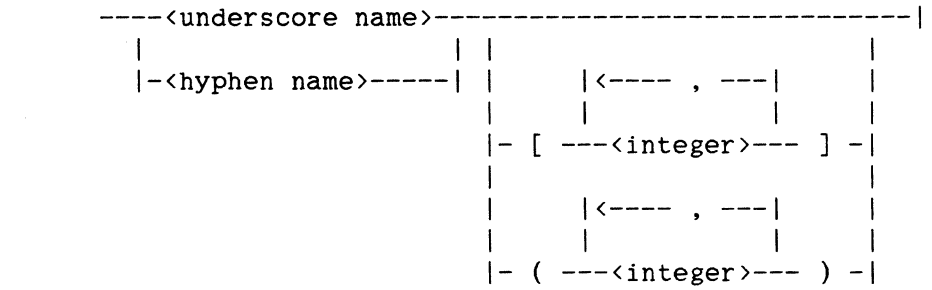
Required Parameters:	Data Type of Parameters:
<structure>	DISPLAY
<item name>	DISPLAY
<variable>	DISPLAY or KANJI

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

A COBOL/COBOL74 variable that holds the string from the data base and may be defined as PIC X(2n) or PIC X(n) USAGE IS KANJI.

Example

Declaration of Variables:

```
01  DATA-SET-NAME        PIC X(17) VALUE IS "EMP".
01  ITEM-NAME            PIC X(17) VALUE IS SPACES.
01  VAR-1                PIC X(10) USAGE IS KANJI,
                                   VALUE IS SPACES.
```

Invocation of Entry Points:

```
MOVE "EMP-KNAME" TO ITEM-NAME.
CALL "DBPUTKANJI OF DMINTERPRETER"
    USING DATA-SET-NAME,ITEM-NAME,VAR-1
    GIVING RESULT.
```

DBPUTREAL

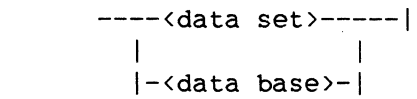
DBPUTREAL puts a real value into a particular data item in the current record of the specified structure. For COBOL, the parameter for the real value must be declared in the working storage section of the program as a level 77 COMP-4 item. For COBOL74, the parameter for the real value must be declared in the working storage section of the program as a level 77 REAL item. The item must have been declared in the DASDL as a NUMBER, REAL, or FIELD item. If the parameter for the real value produces an incorrect new value, an exception is returned.

If the number of significant digits in an item exceeds the capacity of a real variable, a COBOL program should call the DBPUTDOUBLE entry point to put real values into items of this type.

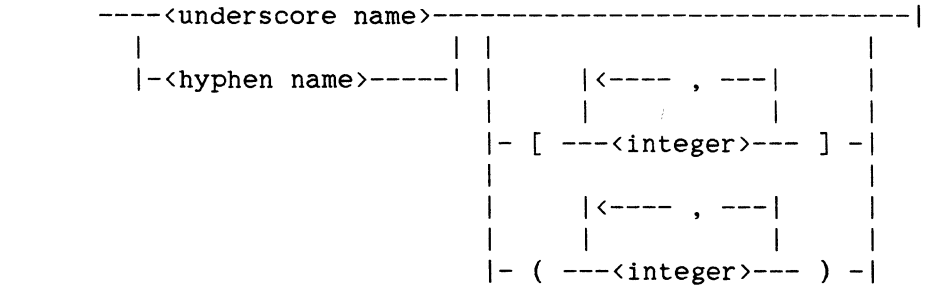
Required Parameters:	Data Type of Parameters:
<structure>	DISPLAY
<item name>	DISPLAY
<variable>	COMP-4 (for COBOL) or REAL (for COBOL74)

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

<variable>

A COBOL real variable.

Example

The following program fragments show how DBPUTREAL is used to move a data item value from a real variable declared by the programmer to a DMSII user work area.

Declaration of Variables:

```
01  DATA-SET-NAME        PIC X(17) VALUE IS "EMP".
01  ITEM-NAME            PIC X(17) VALUE IS SPACES.
77  VAR-2                COMP-4.
```

Invocation of Entry Point:

```
MOVE "EMP-NO" TO ITEM-NAME.
MOVE  11  TO VAR-2.
CALL "DBPUTREAL OF DMINTERPRETER"
    USING DATA-SET-NAME,ITEM-NAME,VAR-2
    GIVING RESULT.
```

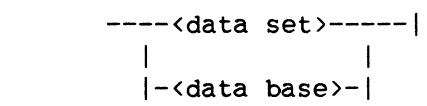
DBPUTBOOLEAN

DBPUTBOOLEAN puts a Boolean value into a particular BOOLEAN data item. The parameter used to carry the BOOLEAN value must be a DISPLAY variable containing the string "TRUE" or "FALSE".

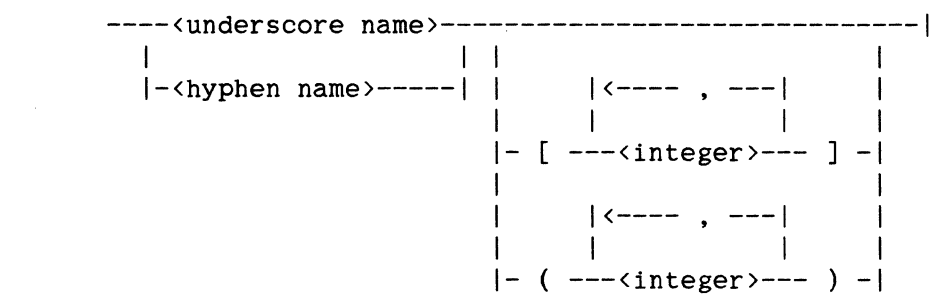
Required Parameters:	Data Type of Parameters:
<structure>	DISPLAY
<item name>	DISPLAY
<variable>	DISPLAY

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

<variable>

A COBOL DISPLAY variable that holds the string "TRUE" or the string "FALSE".

Example

The following program fragments show how DBPUTBOOLEAN is used to move a data item value of "TRUE" or "FALSE" from a COBOL program to a user work area.

Declaration of Variables:

```
01  DATA-SET-NAME        PIC X(17) VALUE IS "EMP".
01  ITEM-NAME            PIC X(17) VALUE IS SPACES.
01  VAR-1                PIC X(10) VALUE IS SPACES.
```

Invocation of Entry Point:

```
MOVE "EMP-SEC-CLEAR" TO ITEM-NAME.
MOVE "TRUE" TO VAR-1.
CALL "DBPUTBOOLEAN  OF DMINTERPRETER"
    USING DATA-SET-NAME,ITEM-NAME,VAR-1
    GIVING RESULT.
```

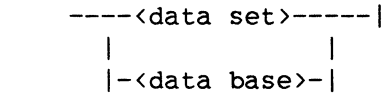
DBPUTDOUBLE

DBPUTDOUBLE puts a double-precision real value into a data item in the current record of the specified structure. The data item must be a NUMBER, REAL, or FIELD item. If the variable produces an incorrect new value, an exception is returned.

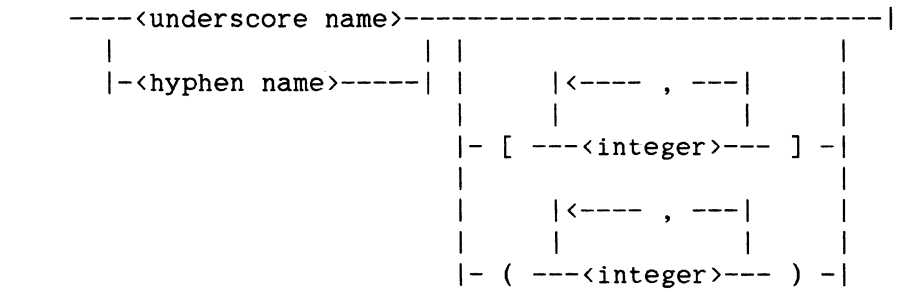
Required Parameters:	Data Type of Parameters:
<structure>	DISPLAY
<item name>	DISPLAY
<variable>	COMP-5 (for COBOL)
	or
	DOUBLE (for COBOL74)

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

<variable>

A COBOL variable that can hold a double-precision numeric value.

Example

The following program fragments show how DBPUTDOUBLE is used to move a data item value from a numeric variable declared by the programmer to a DMSII user work area.

Declaration of Variables:

```
01  DATA-SET-NAME        PIC X(17) VALUE IS "EMP".
01  ITEM-NAME            PIC X(17) VALUE IS SPACES.
77  VAR-3                COMP-5.
```

Invocation of Entry Point:

```
MOVE "EMP-SSN" TO ITEM-NAME.
MOVE  573230911 TO VAR-3.
CALL "DBPUTDOUBLE  OF DMINTERPRETER"
     USING DATA-SET-NAME,ITEM-NAME,VAR-3
     GIVING RESULT.
```

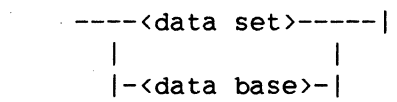
DBPUTNULL

DBPUTNULL sets a particular data item in the current record of the specified structure to its NULL value. The data item chosen should not be of type BOOLEAN, COUNT, RECORDTYPE, or POPULATION.

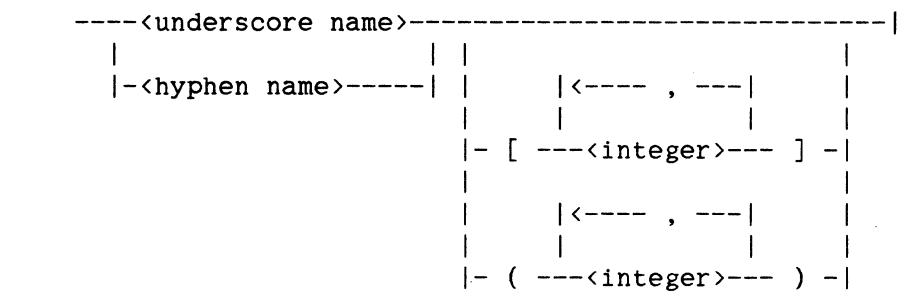
Required Parameters:	Data Type of Parameters:
<structure>	DISPLAY
<item name>	DISPLAY

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

Example

The following program fragments show how DBPUTNULL is used to set a data base item to its NULL value.

Declaration of Variables:

```
01  DATA-SET-NAME        PIC X(17) VALUE IS "EMP".
01  ITEM-NAME            PIC X(17) VALUE IS SPACES.
```

Invocation of Entry Point:

```
MOVE "EMP-SALARY" TO ITEM-1.
CALL "DBPUTNULL OF DMINTERPRETER"
    USING DATA-SET-NAME,ITEM-1
    GIVING RESULT.
```

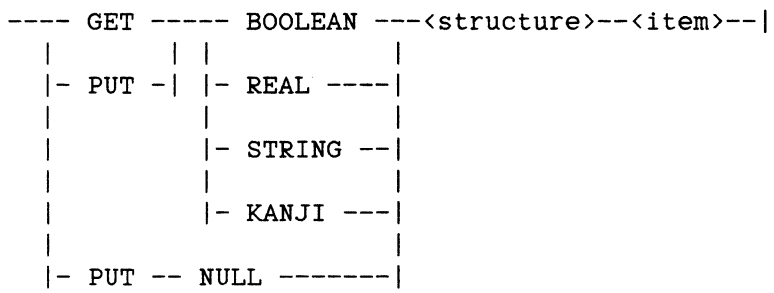
DBDATA

DBDATA allows "GET" and "PUT" commands to be dynamically constructed during program execution. The data request parameter contains a "GET" or "PUT" operation. The data parameter receives the value for "GET" operations and is used to hold the new value for "PUT" operations.

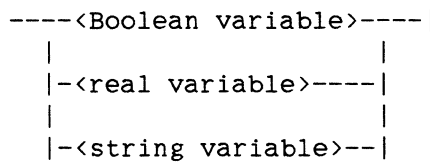
Required Parameters:	Data Type of Parameters:
<data request>	DISPLAY
<data>	DISPLAY

Syntax

<data request>



<data>



Explanation

GET <data type> <structure> <item>

Specifies that a data value of type BOOLEAN, REAL, STRING, or KANJI will be moved from an item in a specified data base structure to a variable declared by the programmer.

PUT <data type> <structure> <item>

Specifies that a data value of type BOOLEAN, REAL, STRING, KANJI, or NULL will be moved from a variable declared by the programmer to an item in a specified data base structure.

<data>

For PUT operations, this parameter holds data that will be moved to the data base. For GET operations, this parameter receives data from the data base.

Example

The following program fragments show how DBDATA is used to move data from a COBOL program to the data base.

Declaration of Variables:

```
01  DATA-BUFFER          PIC X(10) VALUE IS SPACES.
01  DATA-REQUEST         PIC X(40) VALUE IS SPACES.
```

Invocation of Entry Point:

```
MOVE "GET REAL EMP EMP-NO" TO DATA-REQUEST.
CALL "DBDATA OF DMINTERPRETER"
    USING DATA-REQUEST,DATA-BUFFER
    GIVING RESULT.
```

3.3 EXCEPTION-HANDLING ENTRY POINTS

The exception-handling entry points are among the functions of the interpretive interface that are allowed even when the data base is closed.

All COBOL/COBOL74 entry points of the DMINTERPRETER Library (except DBEXCEPTIONTEXT and DBEXCEPTIONNAME) return a particular value to indicate that an exception has occurred. After a COBOL or COBOL74 entry point call to the DMINTERPRETER Library has been completed, the entry point returns an integer to the variable provided in the GIVING clause in the COBOL or COBOL74 program. The returned value is 1 if an exception has occurred and 0 if no exception has occurred.

The DBSTATUS entry point must be subsequently called to retrieve, through its exception parameter, the actual exception word for the last entry point called. This exception word can be passed to DBEXCEPTIONNAME and/or DBEXCEPTIONTEXT to find out the name of the exception or the full exception message, respectively.

When the DBEXCEPTIONTEXT or DBEXCEPTIONNAME entry point is called, the DMINTERPRETER Library calls the DMSUPPORT Library to retrieve the text or the name of the exception message. The DBEXCEPTIONTEXT and DBEXCEPTIONNAME entry points can return any exception that can be returned by the DMSII user language interface. In addition, the DMSUPPORT Library contains a special category of exceptions called INTLIBERRORS, for errors peculiar to the interpretive interface. (Refer to the appendix "DMSII Exceptions and Errors" for a list of exception categories and subcategories and the corresponding exception messages.)

DBSTATUS

DBSTATUS returns the exception word of the last entry point call that was made. The COBOL program does not need to directly examine the contents of this exception, but should pass it on to the DBEXCEPTIONTEXT or DBEXCEPTIONNAME entry point, which use the contents of the exception variable to look up the exception message or exception name found in the DMSUPPORT Library.

Required Parameters:		Data Type of Parameters:
COBOL:	<exception>	COMP-2
COBOL74:	<exception>	COMP

Explanation

<exception>

A variable declared at level 01, PIC 9(12) as a COMP-2 or COMP item.

Example

The following program fragments show how DBSTATUS is used to obtain the exception result from the last entry point call that was made.

Declaration of Variables for COBOL:

```
01  EXCEPTION-WORD        PIC 9(12) COMP-2.
```

Declaration of Variables for COBOL74:

```
01  EXCEPTION-WORD        PIC 9(12) COMP.
```

Invocation of Entry Point for COBOL and COBOL74:

```
CALL "DBSTATUS OF DMINTERPRETER"
    USING EXCEPTION-WORD.
```

DBEXCEPTIONTEXT

After DBSTATUS returns the exception-word for the latest call on DMINTERPRETER, DBEXCEPTIONTEXT can return the text of the exception message in its text parameter. The text parameter contains "NO EXCEPTION" if no exception has occurred. If text is not long enough to hold the entire message, the message is truncated to fit in the available space. If the space is longer than the message text, then it is padded with trailing blanks.

A call on DBEXCEPTIONTEXT should be preceded by a call on DBSTATUS. DBEXCEPTIONTEXT returns a message that can later be displayed by the program. If an error message regarding the DMSUPPORT Library is encountered when DBEXCEPTIONTEXT is called, the user should investigate whether the DMSUPPORT Library is currently unavailable, possibly for security reasons.

Required Parameters:		Data Type of Parameters:
COBOL:	<exception>	COMP-2
	<text>	DISPLAY
COBOL74:	<exception>	COMP
	<text>	DISPLAY

Explanation

<exception>

A variable declared at level 01, PIC 9(12) as a COMP-2 or COMP item, containing the exception information from a DBSTATUS entry point call.

<text>

A variable declared as a DISPLAY item, with enough room to hold error messages without truncation if possible.

Example

The following program fragments show DBEXCEPTIONTEXT is used to obtain the exception message that corresponds to the exception result returned by the DBSTATUS entry point.

Declaration of Variables for COBOL:

```
01  EXCEPTION-WORD       PIC 9(12) COMP-2.
01  EXCEPTION-MSG        PIC X(80).
```

Declaration of Variables for COBOL74:

```
01  EXCEPTION-WORD       PIC 9(12) COMP.
01  EXCEPTION-MSG        PIC X(80).
```

Invocation of Entry Point for COBOL and COBOL74:

```
CALL "DBSTATUS OF DMINTERPRETER"
    USING EXCEPTION-WORD.
CALL "DBEXCEPTIONTEXT OF DMINTERPRETER"
    USING EXCEPTION-WORD,EXCEPTION-MSG.
MOVE EXCEPTION-MSG TO REMOTE-REC.
    WRITE REMOTE-REC.
```

DBEXCEPTIONNAME

For an exception returned by DBSTATUS, DBEXCEPTIONNAME returns in the text parameter the name of the type of exception that occurred. The text contains either the name of a DMSII exception category, such as NOTFOUND or OPENERROR, or "NO EXCEPTION" if no exception has occurred. If text is not long enough to hold the entire message, the message is truncated to fit in the available space. If text is longer than the message, the extra space is padded with trailing blanks.

A call on DBEXCEPTIONNAME should be preceded by a call on DBSTATUS. If an error message regarding the DMSUPPORT Library is encountered when DBEXCEPTIONNAME is called, the user should investigate whether the DMSUPPORT Library is currently unavailable, possibly for security reasons.

Required Parameters:		Data Type of Parameters:
COBOL:	<exception>	COMP-2
	<text>	DISPLAY
COBOL74:	<exception>	COMP
	<text>	DISPLAY

Explanation

<exception>

A variable declared at level 01, PIC 9(12) as a COMP-2 or COMP item, containing the exception information from a DBSTATUS entry point call.

<text>

A variable declared as a DISPLAY item, with enough room to hold error messages without truncation if possible.

Example

The following program fragments show how DBEXCEPTIONNAME is used to obtain the name of the exception that corresponds to the exception result returned by the DBSTATUS entry point.

Declaration of Variables for COBOL:

```
01  EXCEPTION-WORD       PIC 9(12) COMP-2.
01  EXCEPTION-NAME       PIC X(14).
```

Declaration of Variables for COBOL74:

```
01  EXCEPTION-WORD       PIC 9(12) COMP.
01  EXCEPTION-NAME       PIC X(14).
```

Invocation of Entry Point for COBOL and COBOL74:

```
CALL "DBSTATUS OF DMINTERPRETER"
    USING EXCEPTION-WORD.
CALL "DBEXCEPTIONNAME OF DMINTERPRETER"
    USING EXCEPTION-WORD,EXCEPTION-NAME.
MOVE EXCEPTION-NAME TO REMOTE-REC.
    WRITE REMOTE-REC.
```

3.4 ATTRIBUTE-SETTING ENTRY POINTS

DBSETLIMIT, the only COBOL/COBOL74 attribute-setting entry point, is used to restrict the number of times the DMINTERPRETER can call the Accessroutines to complete a find. FINDLIMIT is the only valid limit type allowed for the DBSETLIMIT entry point.

DBSETLIMIT can be used to declare DMINTERPRETER entry points in application programs, unless renaming is desired or necessary due to limitations of a particular language.

DBSETLIMIT

DBSETLIMIT sets the various limit values for the DMINTERPRETER.

	Required Parameters:	Data Type of Parameters:
COBOL:	<limittype>	DISPLAY
	<limitvalue>	DISPLAY
COBOL74:	<limittype>	DISPLAY
	<limitvalue>	DISPLAY

Explanation

<limittype>=FINDLIMIT

A string parameter that specifies the type of limit to be set, and specifies the maximum number of times the DMINTERPRETER can call the Accessroutines to perform a FIND or LOCK.

FINDLIMIT is the only <limittype> currently defined. By setting FINDLIMIT the user can prevent long linear searches.

If an invalid <limittype> is specified, INTLIBERROR 33 is returned. If the DMINTERPRETER exceeds the FINDLIMIT while performing a FIND or LOCK, INTLIBERROR 35 is returned.

<limitvalue>

An integer string parameter which specifies the value to be assigned to <limittype>. Negative values are not allowed for <limittype>. Specifying a <limitvalue> of zero disables the FINDLIMIT mechanism, allowing infinitely long linear searches.

When the DMINTERPRETER is first initialized, FINDLIMIT is set to the default value of zero.

4 ENTRY POINTS IN ALGOL

This section explains the ALGOL interface to the DMINTERPRETER Library, and provides a sample application program written in ALGOL to illustrate the following concepts:

1. How ALGOL entry points of the DMINTERPRETER Library are declared in an ALGOL application program.

2. How an ALGOL application program "calls" or "invokes" entry points of the DMINTERPRETER Library.

3. How the DMINTERPRETER Library returns results to ALGOL programs.

4. How exceptions returned by entry point calls are handled.

This section also lists all entry points of the DMINTERPRETER Library for ALGOL and describes their functions and parameters. The following information is included under each description of an entry point:

1. A brief explanation of the function of the entry point.

2. Syntax diagrams and semantics that describe required parameters. (See the appendix "Syntax Diagrams for Entry Point Parameters" for more detailed syntax diagrams for data base constructs used with the interpretive interface.)

3. A program fragment that illustrates local declaration of the entry point in an application program.

4. A program fragment that illustrates the assignment of values to variables used as parameters.

5. A program fragment that illustrates the invocation of the entry point.

Table 4-1 summarizes the ALGOL entry points, and lists them in the order in which they are described later in this section.

Table 4-1. Summary
of ALGOL Entry Points

Standard Entry Points:

ALGOLOPEN
ALGOLCLOSE
ALGOLBEGINTRANSACTION
ALGOLENDTRANSACTION
ALGOLCREATE
ALGOLRECREATE
ALGOLSTORE
ALGOLFREE
ALGOLSET
ALGOLFIND
ALGOLLOCK
ALGOLDELETE
ALGOLVERB

Data Transfer Entry Points:

ALGOLGETSTRING
ALGOLGETKANJI
ALGOLGETREAL
ALGOLGETDOUBLE
ALGOLGETBOOLEAN
ALGOLPUTSTRING
ALGOLPUTKANJI
ALGOLPUTREAL
ALGOLPUTDOUBLE
ALGOLPUTBOOLEAN
ALGOLPUTNULL
ALGOLDATA

Exception Handling Entry Points:

ALGOLSTATUS
ALGOLEXCEPTIONTEXT
ALGOLEXCEPTIONNAME

Attribute Setting Entry Point:

ALGOLSETLIMIT

LIBRARY DECLARATION

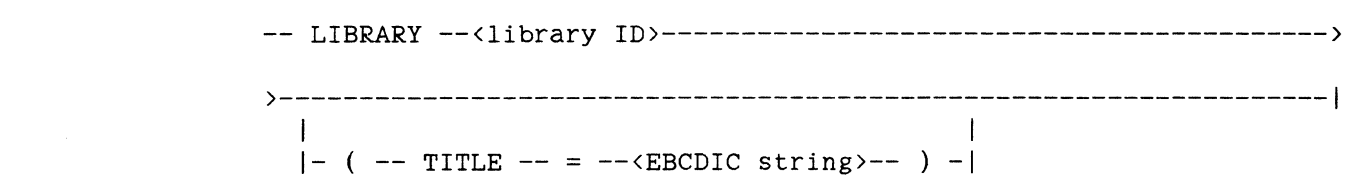
The first step in writing an application program that accesses the interpretive interface is to provide for linkage of entry points declared in an application program to entry points of the DMINTERPRETER Library. Linkage to the DMINTERPRETER Library must be direct, because the library actually contains the procedures that are named in its EXPORT declaration. (For more information, refer to the ALGOL Programming Reference Manual, Volume 1: Basic Implementation.)

In ALGOL, linkage to the DMINTERPRETER Library is accomplished by declaring the library, and then assigning the external name of the library code file (which has been compiled for a particular data base) to the library identifier by using the TITLE attribute. The library identifier is used as an internal name within the application program to refer to the DMINTERPRETER Library.

The following diagrams show how library declaration should be done in ALGOL programs that are linking to entry points of the DMINTERPRETER Library.

Syntax

<library declaration>



<library ID>



Explanation

The library declaration declares an identifier to be a library ID, and specifies values for the library attributes associated with the library. In the TITLE clause, the EBCDIC string must have a period as its last character and must be a properly formed file title. If the TITLE clause is used in the library declaration, linkage to the DMINTERPRETER Library is complete.

If the short form of library declaration (that is, "LIBRARY library ID") is used, the title of the library code file that has been compiled for a particular data base must be specified in the following manner:

```
<library ID>.TITLE := "EBCDIC string";
```

In the sample ALGOL program provided in the appendix "Sample Programs", the short form of library declaration has been used. The library ID is DMI, and the title of the library code file is specified as follows:

```
DMI.TITLE:= "DMINTERPRETER/EMPJOB";
```

Libraries may be declared in any block of a user program. The library and its entry points are valid within the scope of the block; upon exiting the block, linkage to the library is broken and the count of users is decremented.

ENTRY POINT DECLARATION

To ease the process of writing programs that use the DMINTERPRETER Library, the DATABASE/DMINTERPRETER symbolic file provided on the release tapes contains a declaration of the library and its entry points for the ALGOL language. This declaration can be included in application programs by using the $ INCLUDE option for the following sequence range within DATABASE/DMINTERPRETER:

20100000 - 20199999

The library identifier is DMI, and the includable entry points rename exported entry points as follows:

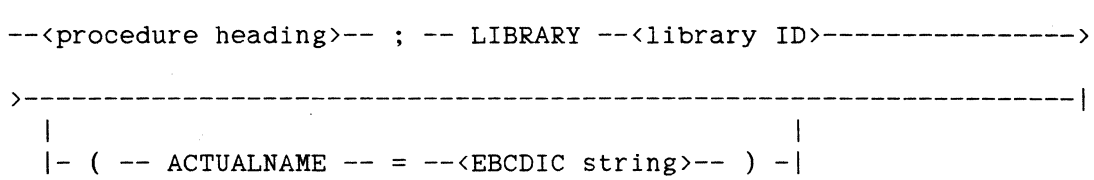
Name of Exported Entry Point:	Renamed As:
ALGOLOPEN	DBOPEN
ALGOLCLOSE	DBCLOSE
ALGOLBEGINTRANSACTION	DBBEGINTRANSACTION
ALGOLENDTRANSACTION	DBENDTRANSACTION
ALGOLCREATE	DBCREATE
ALGOLRECREATE	DBRECREATE
ALGOLSTORE	DBSTORE
ALGOLFREE	DBFREE
ALGOLSET	DBSET
ALGOLFIND	DBFIND
ALGOLLOCK	DBLOCK
ALGOLDELETE	DBDELETE
ALGOLVERB	DBVERB
ALGOLGETSTRING	DBGETSTRING
ALGOLGETKANJI	DBGETKANJI
ALGOLGETREAL	DBGETREAL
ALGOLGETDOUBLE	DBGETDOUBLE
ALGOLGETBOOLEAN	DBGETBOOLEAN
ALGOLPUTSTRING	DBPUTSTRING
ALGOLPUTKANJI	DBPUTKANJI
ALGOLPUTREAL	DBPUTREAL
ALGOLPUTDOUBLE	DBPUTDOUBLE
ALGOLPUTBOOLEAN	DBPUTBOOLEAN
ALGOLPUTNULL	DBPUTNULL
ALGOLDATA	DBDATA
ALGOLSTATUS	DBSTATUS
ALGOLEXCEPTIONTEXT	DBEXCEPTIONTEXT
ALGOLEXCEPTIONNAME	DBEXCEPTIONNAME
ALGOLSETLIMIT	DBSETLIMIT

Data types for the parameters of each entry point are given under individual entry point descriptions later in this section. The includable sequence range declares all entry points, renaming them with the "DB" prefix. Users who want to rename the exported ALGOL entry points in a different manner, or declare only a subset of the entry points, must individually declare the entry points themselves.

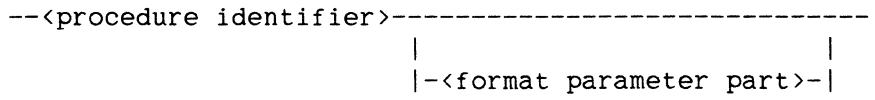
The following diagram shows the declaration of a library entry point in ALGOL:

Syntax

<library entry point declaration>



<procedure heading>



Explanation

The library entry point declaration declares a procedure to be an entry point in the library known to this program through the library ID. This procedure cannot be declared FORWARD or EXTERNAL.

The ACTUALNAME clause allows the renaming of a library entry point by using a different procedure identifier in the procedure heading which can still link to the appropriate entry point through ACTUALNAME. The EBCDIC string in the "ACTUALNAME =" clause cannot contain any leading, trailing, or embedded blanks and must be an ALGOL valid identifier. (An ALGOL valid identifier is defined as any sequence of characters beginning with a letter and consisting of letters, digits, hyphens, and underscores.) The EBCDIC string is the actual name that is given for this procedure in the template of this library.

Example

The following example shows how an entry point may be declared within an ALGOL program.

Library Entry Point Declaration:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBOPEN (OPENTYPE);
   STRING OPENTYPE;
   LIBRARY DMI (ACTUALNAME = "ALGOLOPEN");
```

INVOCATION OF ENTRY POINTS

In ALGOL, each entry point of the DMINTERPRETER Library is a Boolean procedure. The following is a typical method of invoking an entry point of the DMINTERPRETER Library, where RSLT is a Boolean variable, and EXCEPTIONHANDLER is a procedure for handling exceptions:

How to Invoke an Entry Point:

```
RSLT:= DBOPEN(OPEN_TYPE) ;
IF RSLT
   THEN EXCEPTIONHANDLER;
```

After an entry point call on the DMINTERPRETER Library, a result descriptor is returned as a Boolean value. This result descriptor is a 48-bit word which is the standard Data Management System II (DMSII) exception word; it includes the exception category and subcategory, and the structure number involved. When a call on the DMINTERPRETER Library causes an exception, the last bit in the 48-bit word is set to 1. (When a call on the library is successful, the 48-bit word contains all zeros.) For more information on exceptions, refer to "Exception-Handling Entry Points" later in this section.

4.1 STANDARD ENTRY POINTS

Standard entry points correspond to functions performed by DMSII user language interface statements. The results returned by standard entry points are used for exception handling.

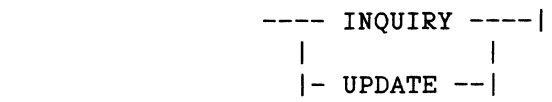
ALGOLOPEN

ALGOLOPEN opens the data base for inquiry or update, depending on which open disposition is used. ALGOLOPEN must be called before any data base operations are performed. ALGOLOPEN, the exception handling entry points (ALGOLSTATUS, ALGOLEXCEPTIONTEXT, and ALGOLEXCEPTIONNAME), and operations on the intrinsic data set DBSTRUCTURE are the only functions allowed when the data base is closed.

Required Parameters:	Data Type of Parameters:
<open disposition>	STRING

Syntax

<open disposition>



Explanation

INQUIRY

Causes the data base to be opened for INQUIRY. No updates are allowed.

UPDATE

Causes the data base to be opened for UPDATE. Both inquiry and update are allowed.

Example

The following program fragments show how ALGOLOPEN is used to open the data base for UPDATE. The OPEN_TYPE variable must be set to UPDATE or INQUIRY before invoking the entry point.

Local Declaration of Variables:

```
STRING
      OPEN_TYPE
      ;
```

Assignment of Values to Parameters:

```
OPEN_TYPE:="UPDATE";
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBOPEN (OPENTYPE);
   STRING OPENTYPE;
   LIBRARY DMI (ACTUALNAME = "ALGOLOPEN");
```

Invocation of Entry Point:

```
 RSLT:= DBOPEN(OPEN_TYPE) ;
 IF RSLT
    THEN EXCEPTIONHANDLER;
```

ALGOLCLOSE

ALGOLCLOSE explicitly closes the data base. (The data base is implicitly closed when the library execution completes.)

Required Parameters:	Data Type of Parameters:
None	Not applicable

Example

The following example shows how ALGOLCLOSE is used to close the data base. When invoking ALGOLCLOSE, no parameters need to be specified, because none are required for this entry point.

Local Declaration of Variables:

Not applicable.

Assignment of Values to Parameters:

Not applicable.

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBCLOSE;
   LIBRARY DMI (ACTUALNAME = "ALGOLCLOSE");
```

Invocation of Entry Point:

```
RSLT:= DBCLOSE ;
IF RSLT
   THEN EXCEPTIONHANDLER;
```

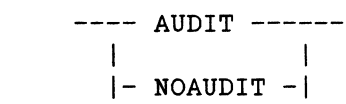
ALGOLBEGINTRANSACTION

ALGOLBEGINTRANSACTION causes the user program to enter transaction state. The audit parameter specifies whether an image of the current restart data set record is to be audited. Calls on ALGOLBEGINTRANSACTION are ignored for unaudited data bases.

Required Parameters:	Data Type of Parameters:
<audit>	STRING

Syntax

<audit>



Explanation

AUDIT

Specifies that an image of the current restart data set will be audited.

NOAUDIT

Specifies that auditing an image of the current restart data set is not required. NOAUDIT is the default value of the audit parameter.

Example

The following example shows how a user program can enter transaction state by calling ALGOLBEGINTRANSACTION. The audit parameter must be set to AUDIT or NOAUDIT before invoking the entry point.

Local Declaration of Variables:

```
STRING
      P_AUDIT
      ;
```

Assignment of Values to Parameters:

```
P_AUDIT:="AUDIT";
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBBEGINTRANSACTION (AUDIT);
   STRING AUDIT;
   LIBRARY DMI (ACTUALNAME = "ALGOLBEGINTRANSACTION")
```

Invocation of Entry Point:

```
RSLT:= DBBEGINTRANSACTION(P_AUDIT) ;
IF RSLT
   THEN EXCEPTIONHANDLER;
```

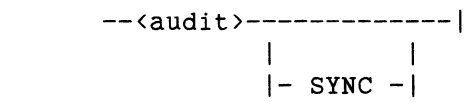
ALGOLENDTRANSACTION

ALGOLENDTRANSACTION causes the user program to leave transaction state. ALGOLENDTRANSACTION should be used only with audited data bases. Calls on ALGOLENDTRANSACTION are ignored for unaudited data bases.

Required Parameters:	Data Types of Parameters:
<auditsync>	STRING

Syntax

<auditsync>



Explanation

<audit>

Specifies whether an image of the current restart data set will be audited (this is the audit parameter discussed under "ALGOLBEGINTRANSACTION" in this section).

<audit> SYNC

Forces a syncpoint, when an image of the current restart data set is audited.

Example

The following program fragments show how ALGOLENDTRANSACTION is used to cause a program to exit transaction state, with the current restart data set record being audited. The P_AUDIT variable must be set to AUDIT or AUDIT SYNC before invoking the entry point.

Local Declaration of Variables:

```
STRING
      P_AUDIT
      ;
```

Assignment of Values to Parameters:

```
P_AUDIT:="AUDIT";
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBENDTRANSACTION (AUDITSYNC);
   STRING AUDITSYNC;
   LIBRARY DMI (ACTUALNAME = "ALGOLENDTRANSACTION");
```

Invocation of Entry Point:

```
RSLT:= DBENDTRANSACTION(P_AUDIT) ;
IF RSLT
   THEN EXCEPTIONHANDLER;
```

ALGOLCREATE

ALGOLCREATE creates a new record for the specified data set. The record type parameter specifies whether the record will be fixed-length or variable-format.

Required Parameters:	Data Types of Parameters:
<data set>	STRING
<record type>	STRING

Syntax

<data set>

```
--<qualified name>--|
```

<record type>

```
-- " ---<integer>--- " --|
      |           |
      |-<blank>---|
```

Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

"<integer>"

A non-zero integer specifies a variable-length record. If the specified integer is 0 (zero), the fixed part of a variable-length record is created.

"<blank>"

If an empty string is specified, a fixed-length record will be created.

Example

The following program fragments show how ALGOLCREATE is used to create a new fixed-length record belonging to a data set named EMP.

Local Declaration of Parameters:

```
STRING
      DATA_SET_NAME
      ,SPACE_1
      ;
```

Assignment of Values to Parameters:

```
DATA_SET_NAME:="EMP";
SPACE_1:=" ";
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBCREATE(STRUCTURE,RECORDTYPE);
   STRING STRUCTURE,RECORDTYPE;
   LIBRARY DMI (ACTUALNAME = "ALGOLCREATE");
```

Invocation of Entry Point:

```
RSLT:= DBCREATE(DATA_SET_NAME,SPACE_1) ;
IF RSLT
   THEN EXCEPTIONHANDLER;
```

ALGOLRECREATE

ALGOLRECREATE re-creates, or copies, a new image of an existing record for the purpose of modification. The data set parameter names the existing data set where the record resides. The record type parameter specifies whether the re-created record will be fixed-length or variable-format.

Required Parameters:	Data Types of Parameters:
<data set>	STRING
<record type>	STRING

Syntax

<data set>

```
--<qualified name>--|
```

<record type>

```
-- " ---<integer>--- " --|
      |           |
      |-<blank>---|
```

Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

"<integer>"

A non-zero integer specifies a variable-length record. If the specified integer is zero, the fixed part of a variable-length record is created.

"<blank>"

If an empty string is specified, a fixed-length record will be created.

Example

The following program fragments show how ALGOLRECREATE is used to re-create an image of an existing fixed-length record belonging to a data set named EMP.

Local Declaration of Variables:

```
STRING
      DATA_SET_NAME
      ,SPACE_1
      ;
```

Assignment of Values to Parameters:

```
DATA_SET_NAME :="EMP";
SPACE_1 :=" ";
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBRECREATE (STRUCTURE,RECORDTYPE);
   STRING STRUCTURE, RECORDTYPE;
   LIBRARY DMI (ACTUALNAME = "ALGOLRECREATE");
```

Invocation of Entry Point:

```
RSLT:= DBRECREATE(DATA_SET_NAME,SPACE_1) ;
IF RSLT
   THEN EXCEPTIONHANDLER;
```

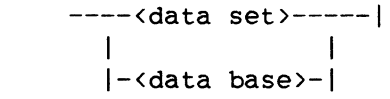
ALGOLSTORE

ALGOLSTORE stores a record in the specified structure. For audited data bases, ALGOLBEGINTRANSACTION must have been called before ALGOLSTORE is called, because the program must be in transaction state in order for a record to be stored. If ALGOLSTORE is called outside of transaction state for an audited data base, an exception is returned.

Required Parameters:	Data Types of Parameters:
<structure>	STRING

Syntax

<structure>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

Example

The following program fragments show how ALGOLSTORE is used to store a record in a data set named EMP.

Local Declaration of Variables:

```
STRING
      DATA_SET_NAME
      ;
```

Assignment of Values to Parameters:

```
DATA_SET_NAME:="EMP";
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBSTORE (STRUCTURE);
   STRING STRUCTURE;
   LIBRARY DMI (ACTUALNAME = "ALGOLSTORE");
```

Invocation of Entry Point:

```
RSLT:= DBSTORE(DATA_SET_NAME) ;
IF RSLT
   THEN EXCEPTIONHANDLER;
```

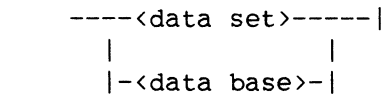
ALGOLFREE

ALGOLFREE unlocks a data set record or a global data record. ALGOLFREE may be used to unlock a record that you anticipate may not be implicitly freed for a long time. After ALGOLFREE is called for a record, other programs may lock that record.

Required Arguments:	Data Types of Arguments:
<structure>	STRING

Syntax

<structure>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

Example

The following program fragments show how ALGOLFREE is used to unlock a record belonging to a specified structure. In this case, a record in the data set EMP is being unlocked.

Local Declaration of Variables:

```
STRING
      DATA_SET_NAME
      ;
BOOLEAN
      RSLT;
```

Assignment of Values to Parameters:

```
DATA_SET_NAME:="EMP";
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBFREE(STRUCTURE);
   STRING STRUCTURE;
   LIBRARY DMI (ACTUALNAME = "ALGOLFREE");
```

Invocation of Entry Point:

```
RSLT:= DBFREE(DATA_SET_NAME) ;
IF RSLT
   THEN EXCEPTIONHANDLER;
```

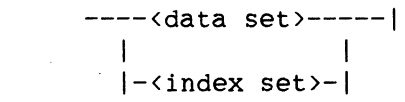
ALGOLSET

ALGOLSET sets particular data sets or index sets to their beginning or ending points, or sets the position pointer of an index set to the current record of a particular data set. The following syntax diagrams and semantics explain which structures can be set to which targets.

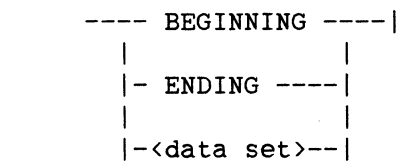
Required Parameters:	Data Type of Parameters:
<structure>	STRING
<target>	STRING

Syntax

<structure>



<target>



Explanation

<data set> BEGINNING

Names a particular data set that will be set to its beginning point at run time.

<data set> ENDING

Names a particular data set that will be set to its ending point at run time.

<index set> <data set>

Names the data set spanned by the index set, and sets the position pointer of the index set to the current record within the spanned data set.

<index set> BEGINNING

Names a particular index set that will be set to its beginning point at run time.

<index set> ENDING

Names a particular index set that will be set to its ending point at run time.

Example

The following program fragments show how ALGOLSET is used to set a particular data base structure to a desired target. In this case, the data set named EMP is set to its beginning point.

Local Declaration of Variables:

```
STRING
      DATA_SET_NAME
      ,BEGIN_1
      ;
```

Assignment of Values to Parameters:

```
DATA_SET_NAME:="EMP";
BEGIN_1:="BEGINNING";
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBSET (STRUCTURE, TARGET);
   STRING STRUCTURE, TARGET;
   LIBRARY DMI (ACTUALNAME = "ALGOLSET");
```

Invocation of Entry Point:

```
RSLT:= DBSET(DATA_SET_NAME,BEGIN_1) ;
IF RSLT
   THEN EXCEPTIONHANDLER;
```

ALGOLFIND

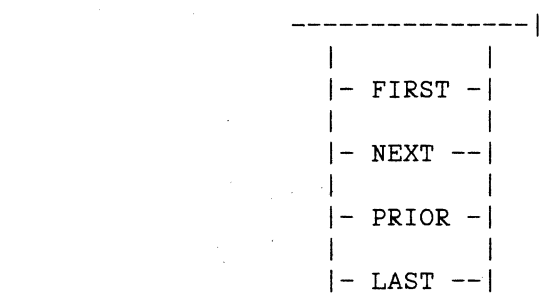
ALGOLFIND transfers a record to the user work area associated with the specified data base structure. The other parameters shown below indicate the direction in which the FIND operation proceeds, and the selection expression used during the search.

An ALGOLFIND call is more sophisticated than a user language interface FIND statement. Several restrictions have been eased, and some algorithmic enhancements have been made. See the appendix "DBFIND Enhancements" for a detailed discussion of these improvements.

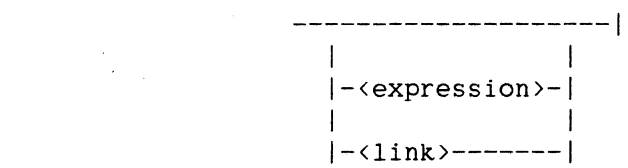
Required Parameters:	Data Type of Parameters:
<direction>	STRING
<structure>	STRING
<condition>	STRING

Syntax

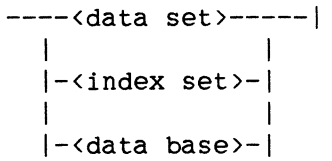
<direction>



<condition>



<structure>



Explanation

(empty)

If the direction parameter is empty, then the FIND operation will transfer the current record in the specified structure to the user work area.

FIRST

Indicates that the FIND operation will transfer the first record in the specified structure to the user work area.

NEXT

Indicates that the FIND operation will transfer the next record in the specified structure to the user work area.

PRIOR

Indicates that the FIND operation will transfer the prior record in the specified structure to the user work area.

LAST

Indicates that the FIND operation will transfer the last record in the specified structure to the user work area.

<data set>

Names a data set in the data base that will be accessed at run time.

<index set>

Names an index set in the data base that will be used for the search.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<expression>

A selection expression used during the DBFIND operation to find records that meet a specified condition.

<link>

Specifies the data base structure or data set item that links the current record to a record in another data set.

Example

The following program fragments show how ALGOLFIND is used to find the first record in a data set named EMP, which satisfies a specified condition.

Local Declaration of Variables:

```
STRING
      DATA_SET_NAME
      ,FIRST_1
      ,CONDITION_1
      ;
```

Assignment of Values to Parameters:

```
DATA_SET_NAME:="EMP";
FIRST_1:="FIRST";
CONDITION_1:="EMP-NO=11";
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBFIND(DIRECTION,
                         STRUCTURE,CONDITION);
   STRING DIRECTION, STRUCTURE, CONDITION;
   LIBRARY DMI (ACTUALNAME = "ALGOLFIND");
```

Invocation of Entry Point:

```
RSLT:= DBFIND(FIRST_1,DATA_SET_NAME,CONDITION_1) ;
IF RSLT
   THEN EXCEPTIONHANDLER;
```

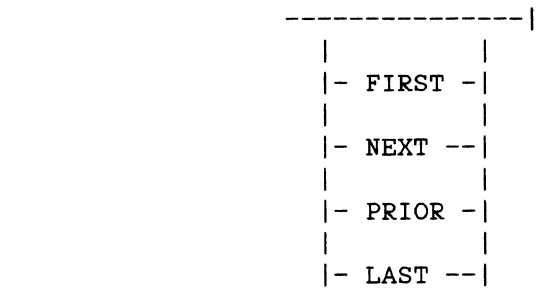
ALGOLLOCK

ALGOLLOCK transfers a record to the user work area associated with the specified data structure and locks it there for modification. The other parameters shown below indicate the direction in which the search for the record proceeds, and the selection expression used during the search.

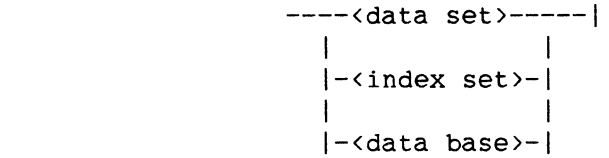
Required Parameters:	Data Type of Parameters:
<direction>	STRING
<structure>	STRING
<condition>	STRING

Syntax

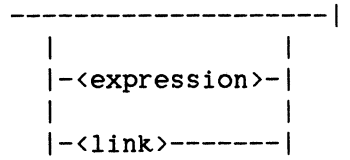
<direction>



<structure>



<condition>



Explanation

(empty)

If the direction parameter is empty, the default value is used, and the FIND operation transfers the current record in the specified structure to the user work area.

FIRST

Indicates that the LOCK operation will transfer the first record in the specified structure to the user work area.

NEXT

Indicates that the LOCK operation will transfer the next record in the specified structure to the user work area.

PRIOR

Indicates that the LOCK operation will transfer the prior record in the specified structure to the user work area.

LAST

Indicates that the LOCK operation will transfer the last record in the specified structure to the user work area.

<data set>

Names a data set in the data base that will be accessed at run time.

<index set>

Names an index set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<expression>

A selection expression used during the ALGOLLOCK operation to lock records that meet a specified condition.

<link>

Specifies the data base structure or data set item that links the current record to a record in another data set.

Example

The following program fragments show how ALGOLLOCK is used to find and lock the first record in a data set named EMP, which satisfies a specified condition.

Local Declaration of Variables:

```
STRING
      DATA_SET_NAME
      ,FIRST_1
      ,CONDITION_1
      ;
```

Assignment of Values to Parameters:

```
DATA_SET_NAME:="EMP";
FIRST_1:="FIRST";
CONDITION_1:="EMP_NO=11";
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBLOCK (DIRECTION,
                          STRUCTURE, CONDITION);
   STRING DIRECTION, STRUCTURE, CONDITION;
   LIBRARY DMI (ACTUALNAME = "ALGOLLOCK");
```

Invocation of Entry Point:

```
RSLT:= DBLOCK(FIRST_1,DATA_SET_NAME,CONDITION_1);
IF RSLT
   THEN EXCEPTIONHANDLER;
```

ALGOLDELETE

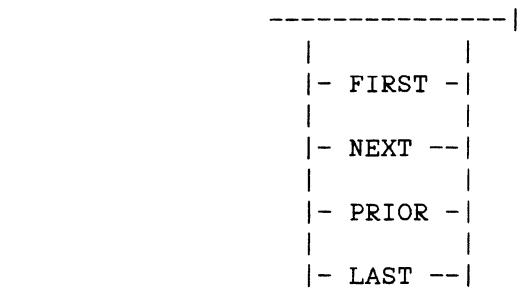
ALGOLDELETE deletes a record from the specified data set. The parameters shown below indicate the direction in which the search and deletion operation proceeds, and the selection expression used during the search.

ALGOLDELETE functions in the same manner as ALGOLFIND, except that a record is deleted after it is found, unless it is a global data record which cannot be deleted. Whenever ALGOLDELETE is called, the data set record is implicitly locked before it is deleted.

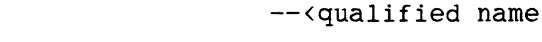
Required Parameters:	Data Type of Parameters:
<direction>	STRING
<data set>	STRING
<condition>	STRING

Syntax

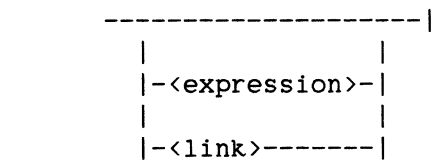
<direction>



<data set>



<condition>



Explanation

(empty)

If the direction parameter is empty, then the ALGOLDELETE call will transfer the current record in the specified structure to the user work area before the record is deleted.

FIRST

Indicates that the ALGOLDELETE call will transfer the first record in the specified structure to the user work area before the record is deleted.

NEXT

Indicates that the ALGOLDELETE call will transfer the next record in the specified structure to the user work area before the record is deleted.

PRIOR

Indicates that the ALGOLDELETE call will transfer the prior record in the specified structure to the user work area before the record is deleted.

LAST

Indicates that the ALGOLDELETE call will transfer the last record in the specified structure to the user work area before the record is deleted.

<qualified name>

See the appendix "Syntax Diagrams for Entry Point Parameters" for basic data base constructs that explain the meaning of qualified name in detail.

<expression>

A selection expression used during the ALGOLDELETE operation to find and delete records that meet a specified condition.

<link>

Specifies the data base structure or data set item that links the current record to a record in another data set.

Example

The following program fragments show how ALGOLDELETE is used to find and delete the first record in a data set named EMP, which satisfies a specified condition.

Local Declaration of Variables:

```
STRING
      DATA_SET_NAME
      ,FIRST_1
      ,CONDITION_1
      ;
```

Assignment of Values to Parameters:

```
FIRST_1 :="FIRST";
DATA_SET_NAME :="EMP";
CONDITION_1 :="EMP-NO=11";
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBDELETE(DIRECTION,
                           STRUCTURE,CONDITION);
   STRING DIRECTION, STRUCTURE, CONDITION;
   LIBRARY DMI (ACTUALNAME = "ALGOLDELETE");
```

Invocation of Entry Point:

```
RSLT:= DBDELETE(FIRST_1,DATA_SET_NAME,CONDITION_1) ;
IF RSLT
   THEN EXCEPTIONHANDLER;
```

ALGOLVERB

ALGOLVERB causes a specified user language interface statement to be performed. The parameters that are used must evaluate to a complete statement. Permissible DMSII user language constructs are shown in the syntax diagrams below, but exception actions and input/output mapping clauses are not allowed. Brief descriptions of these statements are given below. Full descriptions are given in the DMSII User Language Interface Programming Guide.

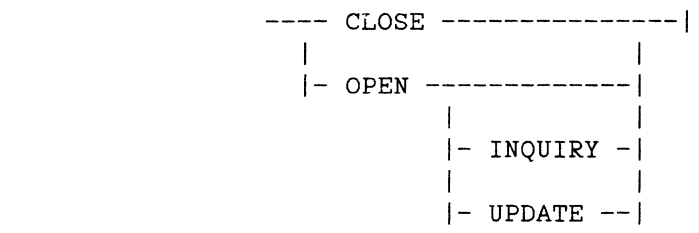
ALGOLVERB allows flexible, run-time definition of DMSII operations, and provides users with an easy transition from DMSII user language interface statements to DMINTERPRETER statements. But because it must completely parse its input, ALGOLVERB runs at higher processor cost than the other, more specialized entry points.

Required Parameters:

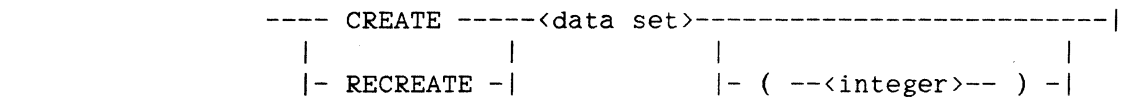
<user language interface statement>

Syntax

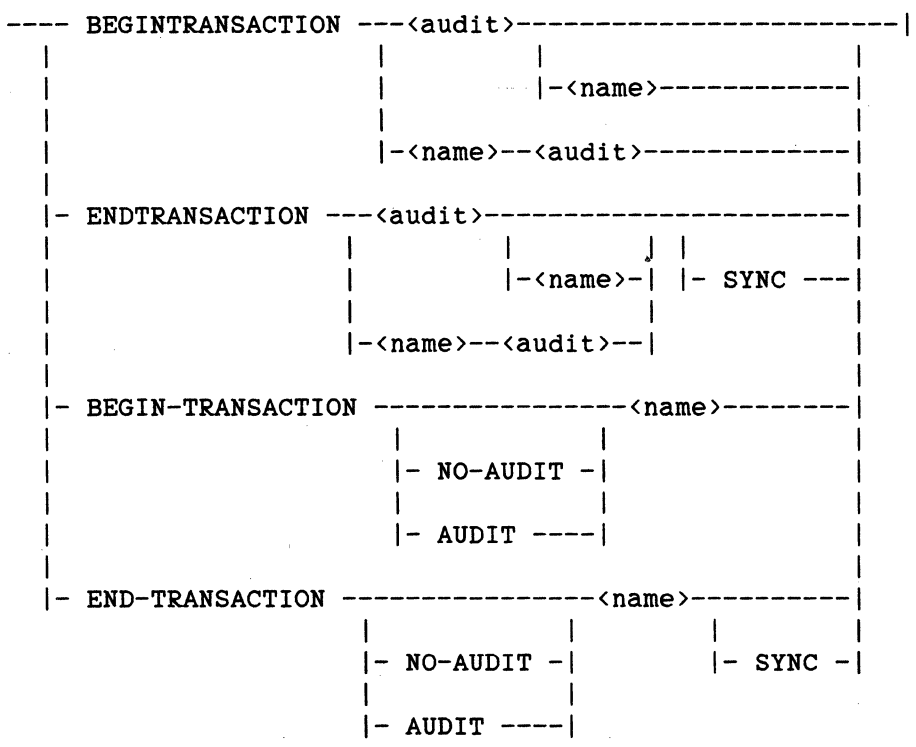
<data base statement>



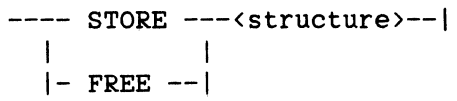
<create statement>



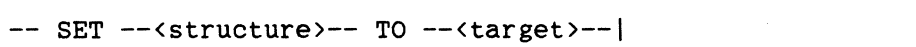
<transaction statement>



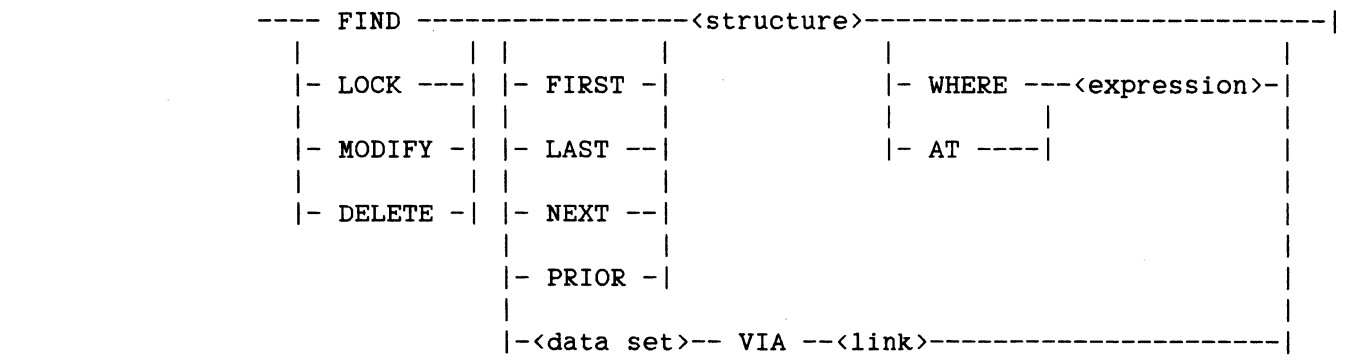
<record statement>



<set statement>



<search statement>



<status statement>

Explanation

<data base statement>

Closes the data base, or opens the data base for inquiry or update.

<create statement>

Initializes the user work area for an existing data set record or a re-created image of a data set record.

<transaction statement>

Causes a program to enter transaction state or exit transaction state, with or without auditing an image of the current restart data set and forcing a syncpoint. The name specifies the restart data set, which is checked for validity when it is present.

<record statement>

Stores a new or modified record in a specified structure, or unlocks a record residing in a specified structure.

<set statement>

Sets a specified structure to its beginning or ending, or to a particular data set if the structure is an index set.

<search statement>

Finds a selected record in a specified structure, and locks the record for modification or deletion.

<status statement>

Allows the user to find out whether a prior data base operation was successful.

Example

The following program fragments show how ALGOLVERB can be called to execute a user language interface statement. When using this entry point, parameters for which values have been initialized in the program are not used. Instead, complete user language interface statements must be used. However, when using the <status statement> construct, just the string "DMSTATUS" is passed as a parameter.

Local Declaration of Variables:

Not applicable when using DBVERB.

Assignment of Values to Parameters:

Not applicable when using DBVERB

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBVERB (HOSTSTATEMENT);
   STRING HOSTSTATEMENT;
   LIBRARY DMI (ACTUALNAME = "ALGOLVERB");
```

Invocation of Entry Point:

```
%
%         USING DBVERB ON A SET STATEMENT
%
    RSLT:= DBVERB("SET EMP TO BEGINNING");
    IF RSLT
       THEN  EXCEPTIONHANDLER ;
%
%         USING DBVERB ON A SEARCH STATEMENT
%
    RSLT:= DBVERB("FIND FIRST EMP AT EMP-NO=11");
    IF RSLT
       THEN EXCEPTIONHANDLER ;
%
%         USING DBVERB ON A STATUS STATEMENT
%
    DBVERB("FIND PRIOR EMP ");
    RSLT:=DBVERB("DMSTATUS");  % TO SEE IF THE FIND WAS OK
```

4.2 DATA TRANSFER ENTRY POINTS

Data transfer entry points, prefixed with "GET" or "PUT," are provided for moving data to and from user work areas associated with specified data base structures. Each item needed from a data set record must be individually moved to or from an ALGOL variable, by calling an appropriate entry point of the DMINTERPRETER Library, before the item can be manipulated by the program.

The "GET" entry points move data from the data base to variables that have been declared by the programmer. The "PUT" entry points move data from ALGOL expressions to the data base.

The DMINTERPRETER Library checks the parameters of data transfer entry points to ensure that the requested operations are compatible with the Data and Structure Definition Language (DASDL) data base description. For example, items declared in the data base description with an OCCURS clause must be subscripted.

The "PUT" entry points (ALGOLPUTDISPLAY, ALGOLPUTREAL, ALGOLPUTBOOLEAN, ALGOLPUTDOUBLE, and ALGOLPUTNULL) return an exception if the value to be put into the data base does not fit into the specified data item. The "GET" entry points (ALGOLGETDISPLAY, ALGOLGETREAL, ALGOLGETDOUBLE, and ALGOLGETBOOLEAN) return an exception if the declared variable is too short to hold the specified data item.

ALGOLGETSTRING

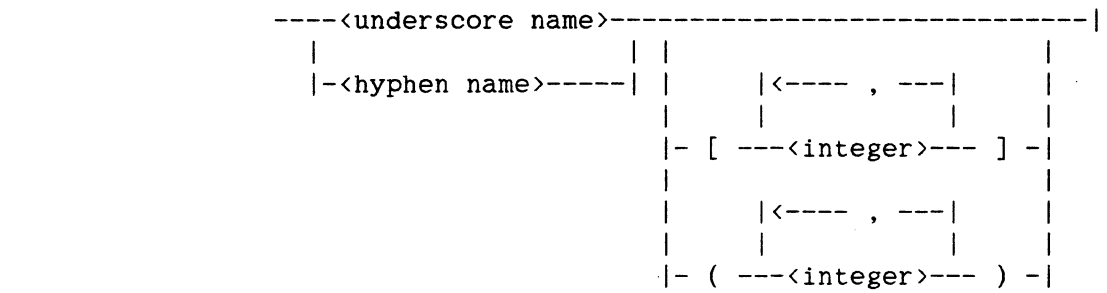
ALGOLGETSTRING sets a variable declared in the application program to a character string representing the contents of an ALPHA item or GROUP item from the current record in the specified structure.

Required Parameters:	Data Type of Parameters:
<structure>	STRING
<item name>	STRING
<variable>	STRING

Syntax

<structure>

<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

An ALGOL string variable.

Example

The following program fragments show how ALGOLGETSTRING is used to move a data item value from the user work area to a variable declared by the programmer. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the value for the ITEM_NAME parameter is assigned in the body of the program.

```
Local Declaration of Variables:

     STRING
          DATA_SET_NAME
          ,ITEM_NAME
          ,EXPR_3
          ;

Assignment of Values to Parameters:

     DATA_SET_NAME:="EMP";
```

Entry Points in ALGOL

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBGETSTRING (STRUCTURE,
                               ITEM,VARIABLE);
   STRING STRUCTURE, ITEM;
   REAL VARIABLE;
   LIBRARY DMI (ACTUALNAME = "ALGOLGETSTRING");
```

Invocation of Entry Point:

```
ITEM_NAME:="EMP-FNAME";
RSLT:= DBGETSTRING(DATA_SET_NAME,ITEM_NAME,EXPR_3);
IF RSLT
   THEN   EXCEPTIONHANDLER  ;
```

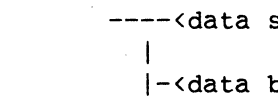
ALGOLGETKANJI

ALGOLGETKANJI sets a variable declared in the application program to a character string representing the contents of an ALPHA item of which USAGE IS KANJI from the current record in the specified structure.

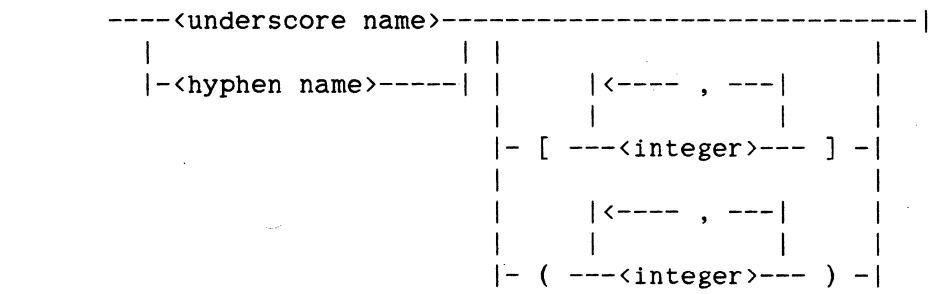
Required Parameters:	Data Type of Parameters:
<structure>	STRING
<item name>	STRING
<variable>	STRING

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

An ALGOL string variable.

Example

The following program fragments show how ALGOLGETKANJI is used to move a data item value from the DMSII user work area to a variable declared by the programmer. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the value for the ITEM_NAME parameter is assigned in the body of the program.

Local Declaration of Variables:

```
STRING
      DATA_SET_NAME
      ,ITEM_NAME
      ,EXPR_3
      ;
```

Assignment of Values to Parameters:

```
DATA_SET_NAME:="EMP";
```

Declaration of Library Entry Points:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBGETKANJI (STRUCTURE,
                              ITEM,VARIABLE);
   STRING STRUCTURE, ITEM, VARIABLE;
   LIBRARY DMI (ACTUALNAME = "ALGOLGETKANJI");
```

Invocation of Entry Points:

```
ITEM_NAME:="EMP-KNAME";
RSLT:= DBGETKANJI(DATA_SET_NAME,ITEM_NAME,EXPR_3);
IF RSLT
   THEN    EXCEPTIONHANDLER  ;
```

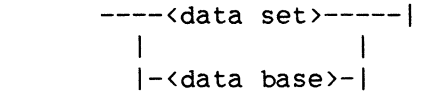
ALGOLGETREAL

ALGOLGETREAL sets a variable declared in the program to the numeric value of a NUMBER, REAL, FIELD, COUNT, RECORDTYPE, or POPULATION data item in the current record of the specified data base structure. If an exception occurs, the variable is set to HIGH-VALUE (all bits ON), and an appropriate exception is returned. Numeric items that are too large to be retrieved by ALGOLGETREAL can be retrieved using the ALGOLGETDOUBLE entry point.

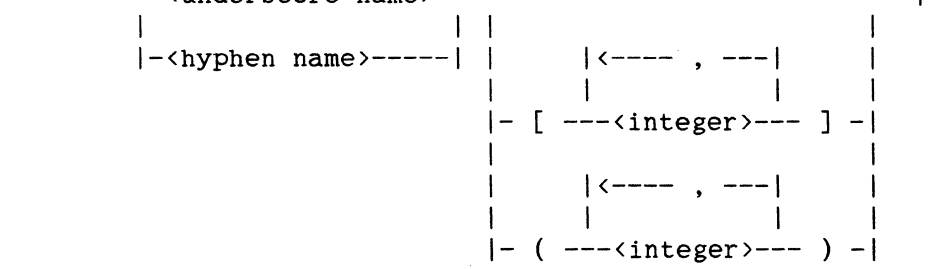
Required Parameters:	Data Type of Parameters:
<structure>	STRING
<item name>	STRING
<variable>	REAL

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

An ALGOL real variable.

Example

The following program fragments show how ALGOLGETREAL is used to move a data item value from a DMSII user work area to a variable declared by the programmer. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the value for the ITEM_NAME parameter is assigned in the body of the program.

Local Declaration of Variables:

```
REAL
    EXPR_1
    ;
STRING
      DATA_SET_NAME
      ,ITEM_NAME
      ;
```

Assignment of Values to Parameters:

```
DATA_SET_NAME:="EMP";
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBGETREAL (STRUCTURE,
                             ITEM,VARIABLE);
   STRING STRUCTURE, ITEM;
   REAL VARIABLE;
   LIBRARY DMI (ACTUALNAME = "ALGOLGETREAL");
```

Invocation of Entry Point:

```
ITEM_NAME:="EMP-NO";
RSLT:= DBGETREAL(DATA_SET_NAME,ITEM_NAME,EXPR_1);
IF RSLT
   THEN   EXCEPTIONHANDLER  ;
```

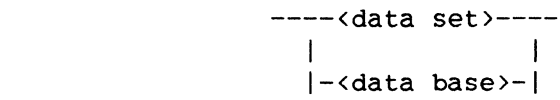
ALGOLGETBOOLEAN

ALGOLGETBOOLEAN retrieves a Boolean value from a BOOLEAN item or a bit in a FIELD item, and sets an ALGOL variable to either the string "TRUE" or the string "FALSE" depending on the value. The ALGOL variable will be set to a false indication if an exception occurs. The character string that is returned will be truncated if the declared variable is too short.

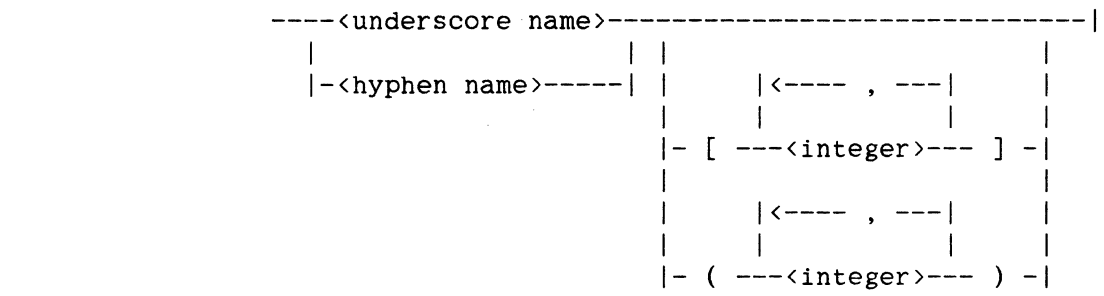
Required Parameters:	Data Type of Parameters:
<structure>	STRING
<item name>	STRING
<variable>	BOOLEAN

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

An ALGOL variable declared as Boolean.

Example

The following program fragments show how ALGOLGETBOOLEAN is used to move a data item value from the user work area to a variable declared by the programmer. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the value for the ITEM_NAME parameter is assigned in the body of the program.

```
Local Declaration of Variables:

     STRING
          DATA_SET_NAME
          ;
     BOOLEAN
          EXPR_4
          ;

Assignment of Values to Parameters:

     DATA_SET_NAME:="EMP";
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBGETBOOLEAN (STRUCTURE,
                                ITEM,VARIABLE);
   STRING STRUCTURE, ITEM;
   BOOLEAN VARIABLE;
   LIBRARY DMI (ACTUALNAME = "ALGOLGETBOOLEAN");
```

Invocation of Entry Point:

```
ITEM_NAME:="EMP-SEC-CLEAR" ;
RSLT:= DBGETBOOLEAN(DATA_SET_NAME,ITEM_NAME,EXPR_4);
IF RSLT
   THEN   EXCEPTIONHANDLER  ;
```

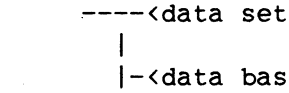
ALGOLGETDOUBLE

ALGOLGETDOUBLE retrieves numeric data items whose number of significant digits exceeds the capacity of a real variable. ALGOLGETDOUBLE sets a variable declared in the program to the numeric value of a NUMBER, REAL, FIELD, COUNT, RECORDTYPE, or POPULATION data item in the current record of the specified structure. If an exception occurs, the declared variable is set to HIGH-VALUE (all BITS on), and an appropriate exception is returned.

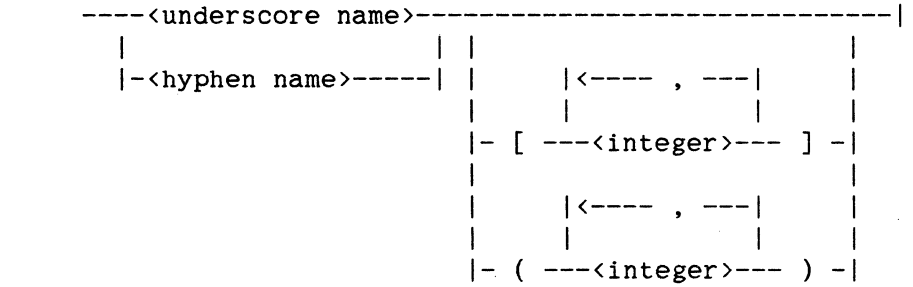
Required Parameters:	Data Type of Parameters:
<structure>	STRING
<item name>	STRING
<variable>	DOUBLE

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

An ALGOL variable declared as DOUBLE.

Example

The following program fragments show how ALGOLGETDOUBLE is used to move a data item value from a DMSII user work area to a variable declared by the programmer. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the value for the ITEM_NAME parameter is assigned in the body of the program.

```
Local Declaration of Variables:

   STRING
         DATA_SET_NAME
         ,ITEM_NAME
         ;
   DOUBLE
         EXPR_2
         ;

Assignment of Values to Parameters:

   DATA_SET_NAME:="EMP";
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBGETDOUBLE (STRUCTURE,
                               ITEM,VARIABLE) ;
   STRING STRUCTURE, ITEM;
   DOUBLE VARIABLE;
   LIBRARY DMI (ACTUALNAME = "ALGOLGETDOUBLE");
```

Invocation of Entry Point:

```
ITEM_NAME:="EMP-SSN" ;
RSLT:= DBGETDOUBLE(DATA_SET_NAME,ITEM_NAME,EXPR_2);
IF RSLT
   THEN   EXCEPTIONHANDLER  ;
```

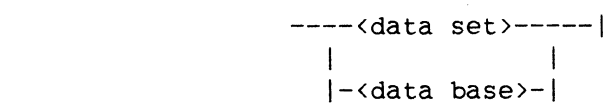
ALGOLPUTSTRING

ALGOLPUTSTRING puts a string into a particular data item in the current record of the specified structure. The data item must be declared in DASDL as an ALPHA or GROUP item. The string must not exceed the length of the data item where it is stored, otherwise an exception is returned.

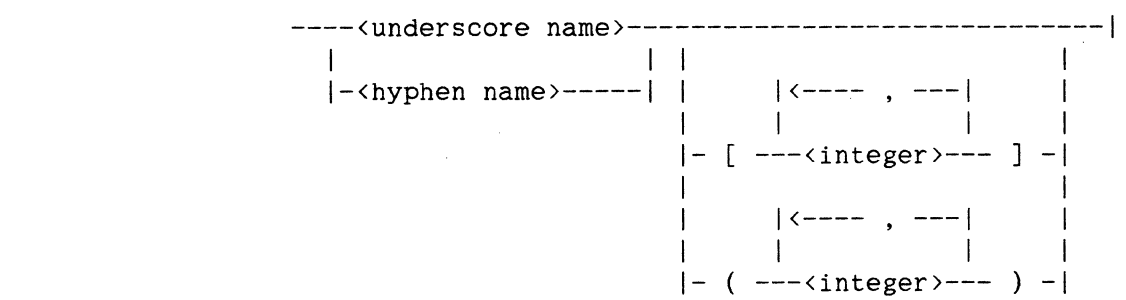
Required Parameters:	Data Type of Parameters:
<structure>	STRING
<item name>	STRING
<string expression>	STRING

Syntax

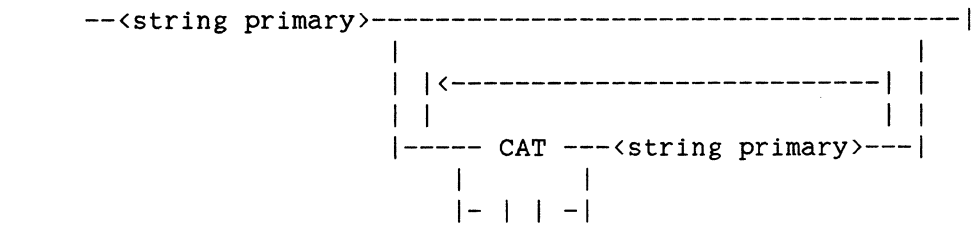
<structure>



<item name>



<string expression>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

<string expression>

An ALGOL string expression that can hold a string to be put into the data base. For more information, refer to the ALGOL Programming Reference Manual, Volume 1: Basic Implementation.

Example

The following program fragments show how ALGOLPUTSTRING is used to move a data item value from a string expression declared by the programmer to a DMSII user work area. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the values for the ITEM_NAME and EXPR_3 parameters are assigned in the body of the program. Concatenation has also been used to create a string for the employee name being put into the data base.

Local Declaration of Variables:

```
STRING
      DATA_SET_NAME
     ,ITEM_NAME
     ,EXPR_3
     ;
```

Assignment of Values to Parameters:

```
DATA_SET_NAME:="EMP";
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBPUTSTRING (STRUCTURE,
                               ITEM,VARIABLE);
   STRING STRUCTURE, ITEM, VARIABLE;
   LIBRARY DMI (ACTUALNAME = "ALGOLPUTSTRING");
```

Invocation of Entry Point:

```
ITEM_NAME:="EMP-FNAME" ;
EXPR_3:="CATHY" ;
RSLT:= DBPUTSTRING(DATA_SET_NAME,ITEM_NAME,
EXPR_3 CAT "LINA" );
IF RSLT
   THEN   EXCEPTIONHANDLER  ;
```

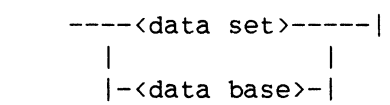
ALGOLPUTKANJI

ALGOLPUTKANJI puts a string into a particular data item in the current record of the specified structure. The data item must be declared in DASDL as an ALPHA item of which USAGE IS KANJI. The string must not exceed the length of the data item where it is stored; otherwise, an exception is returned.

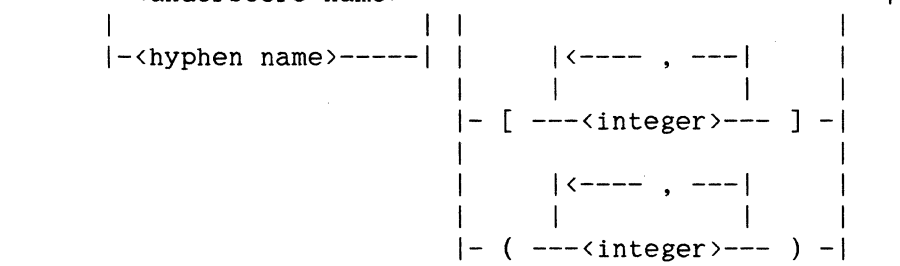
Required Parameters:	Data Type of Parameters:
<structure>	STRING
<item name>	STRING
<variable>	STRING

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

An ALGOL string variable.

Example

The following program fragments show how ALGOLPUTKANJI is used to move a data item value from a string expression declared by the programmer to a DMSII user work area. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the values for the ITEM_NAME and EXPR_3 parameters are assigned in the body of the program. Concatenation has also been used to create a string for the employee name being put into the data base. KANJI_STRING is a string item that contains Kanji characters without the start-of kanji/end-of-kanji (SOK/EOK) characters.

Local Declaration of Variables:

```
STRING
      DATA_SET_NAME
      ,ITEM_NAME
      ,EXPR_3
      ;
```

Assignment of Values to Parameters:

```
DATA_SET_NAME:="EMP";
```

Declaration of Library Entry Points:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBPUTKANJI (STRUCTURE,
                              ITEM,VARIABLE);
   STRING STRUCTURE, ITEM, VARIABLE;
   LIBRARY DMI (ACTUALNAME = "ALGOLPUTKANJI");
```

Invocation of Entry Points:

```
ITEM_NAME:="EMP_KNAME";
EXPR_3:=KANJI_STRING;
RSLT:= DBPUTKANJI(DATA_SET_NAME,ITEM_NAME,
EXPR_3 CAT KANJI_STRING2);
IF RSLT
   THEN    EXCEPTIONHANDLER  ;
```

ALGOLPUTREAL

ALGOLPUTREAL puts a real value into a particular data item in the current record of the specified structure. The item must have been declared in the DASDL as a NUMBER, REAL, or FIELD item. If the parameter for the real value produces an incorrect new value, an exception is returned.

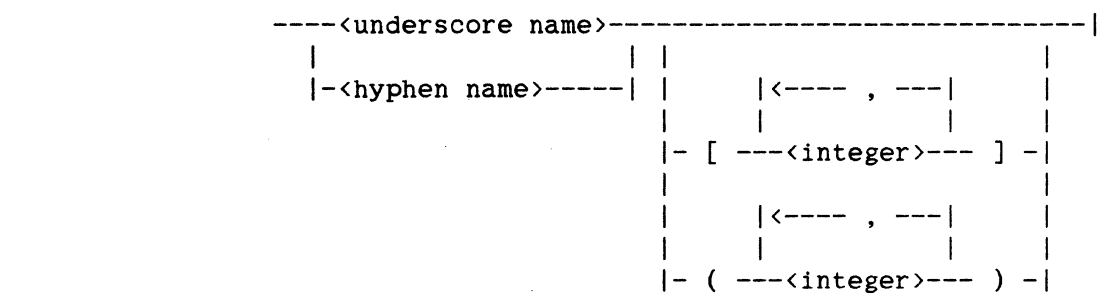
If the number of significant digits in an item exceeds the capacity of a real expression, an ALGOL program should call the ALGOLPUTDOUBLE entry point to put real values into items of this type.

Required Parameters:	Data Type of Parameters:
<structure>	STRING
<item name>	STRING
<arithmetic expression>	REAL

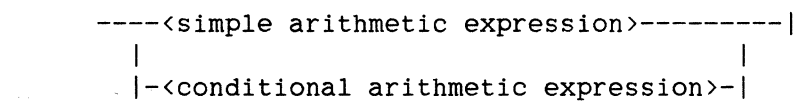
Syntax

<structure>

<item name>



<arithmetic expression>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

<arithmetic expression>

An ALGOL arithmetic expression that can hold a real value to be put into the data base. For more information, refer to the ALGOL Programming Reference Manual, Volume 1: Basic Implementation.

Example

The following program fragments show how DBPUTREAL is used to move a data item value from a real expression declared by the programmer to a DMSII user work area. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the values for the ITEM_NAME and EXPR_1 parameters are assigned in the body of the program.

Local Declaration of Variables:

```
STRING
      DATA_SET_NAME
      ,ITEM_NAME
      ;
REAL
    EXPR_1
    ;
```

Assignment of Values to Parameters:

```
DATA_SET_NAME:="EMP";
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBPUTREAL (STRUCTURE,
                             ITEM,VARIABLE);
   VALUE VARIABLE;
   STRING STRUCTURE, ITEM;
   REAL VARIABLE;
   LIBRARY DMI (ACTUALNAME = "ALGOLPUTREAL");
```

Invocation of Entry Point:

```
ITEM_NAME:= "EMP-NO";
EXPR_1:=11;
RSLT:= DBPUTREAL(DATA_SET_NAME,ITEM_NAME,EXPR_1);
IF RSLT
   THEN   EXCEPTIONHANDLER  ;
```

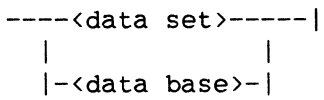
ALGOLPUTDOUBLE

ALGOLPUTDOUBLE puts a double-precision real value into a data item in the current record of the specified structure. The data item must be a NUMBER, REAL, or FIELD item. If the arithmetic expression produces an incorrect new value, an exception is returned.

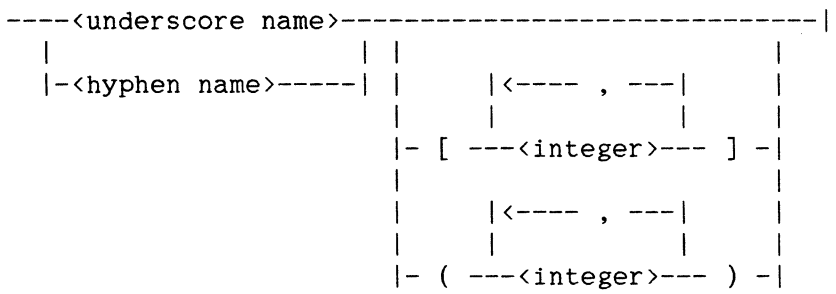
Required Parameters:	Data Type of Parameters:
<structure>	STRING
<item name>	STRING
<arithmetic expression>	DOUBLE

Syntax

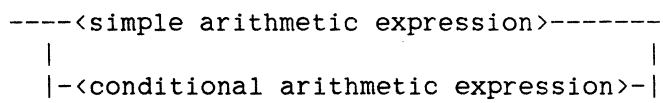
<structure>



<item name>



<arithmetic expression>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

<arithmetic expression>

An ALGOL expression that evaluates to a double-precision numeric value. For more information, refer to the ALGOL Programming Reference Manual, Volume 1: Basic Implementation.

Example

The following program fragments show how ALGOLPUTDOUBLE is used to move a data item value from a numeric variable declared by the programmer to a DMSII user work area. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the values for the ITEM_NAME and EXPR_2 parameters are assigned in the body of the program. The arithmetic expression used is (EXPR_2 - 11).

Local Declaration of Variables:

```
STRING
      DATA_SET_NAME
      ,ITEM_NAME
      ;
DOUBLE
      EXPR_2
      ;
```

Assignment of Values to Parameters:

```
DATA_SET_NAME:="EMP";
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBPUTBOOLEAN (STRUCTURE,
                                ITEM,VARIABLE);
   VALUE VARIABLE;
   STRING STRUCTURE, ITEM;
   DOUBLE VARIABLE;
   LIBRARY DMI (ACTUALNAME = "ALGOLPUTDOUBLE");
```

Invocation of Entry Point:

```
ITEM_NAME:= "EMP-SSN" ;
EXPR_2:=573230911 ;
RSLT:= DBPUTDOUBLE(DATA_SET_NAME,
                   ITEM_NAME,EXPR_2 - 11 );
IF RSLT
   THEN   EXCEPTIONHANDLER  ;
```

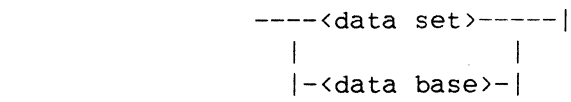
ALGOLPUTBOOLEAN

ALGOLPUTBOOLEAN puts a Boolean value into a particular BOOLEAN item. The parameter used to carry the BOOLEAN value must be a Boolean expression.

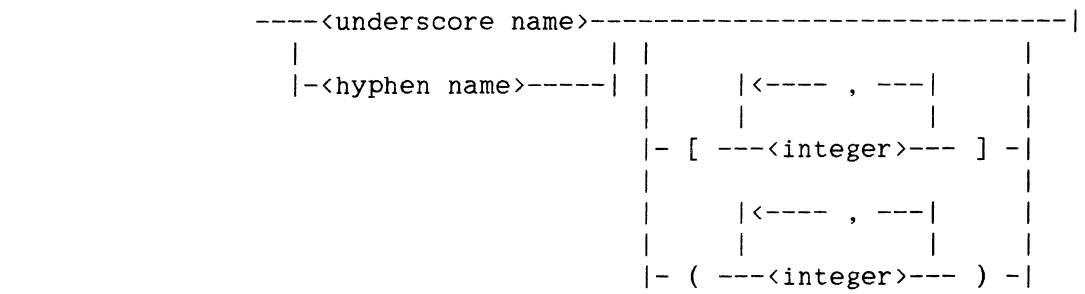
Required Parameters:	Data Type of Parameters:
<structure>	STRING
<item name>	STRING
<Boolean expression>	BOOLEAN

Syntax

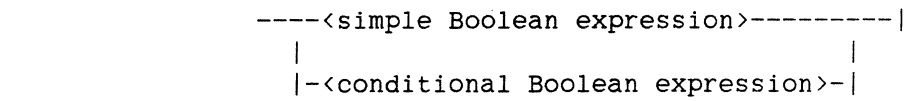
<structure>



<item name>



<Boolean expression>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

<Boolean expression>

An ALGOL expression that evaluates to TRUE or FALSE. For more information, refer to the ALGOL Programming Reference Manual, Volume 1: Basic Implementation.

Example

The following program fragments show how ALGOLPUTBOOLEAN is used to move a Boolean expression evaluating to FALSE from an ALGOL program to a DMSII user work area. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the values for the ITEM_NAME and EXPR_4 parameters are assigned in the body of the program.

Local Declaration of Variables:

```
STRING
      DATA_SET_NAME
      ,ITEM_NAME
      ;
BOOLEAN
      EXPR_4
      ;
```

Assignment of Values to Parameters:

```
DATA_SET_NAME:="EMP";
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBPUTBOOLEAN (STRUCTURE,
                                ITEM,VARIABLE);
   VALUE VARIABLE;
   STRING STRUCTURE, ITEM;
   BOOLEAN VARIABLE;
   LIBRARY DMI (ACTUALNAME = "ALGOLPUTBOOLEAN");
```

Invocation of Entry Point:

```
ITEM_NAME:=  "EMP-SEC-CLEAR" ;
EXPR_4:= TRUE ;
RSLT:= DBPUTBOOLEAN(DATA_SET_NAME,
                    ITEM_NAME,NOT EXPR_4);
IF RSLT
   THEN   EXCEPTIONHANDLER  ;
```

ALGOLPUTNULL

ALGOLPUTNULL sets a particular data item in the current record of the specified structure to its NULL value. The data item chosen should not be of type BOOLEAN, COUNT, RECORDTYPE, or POPULATION.

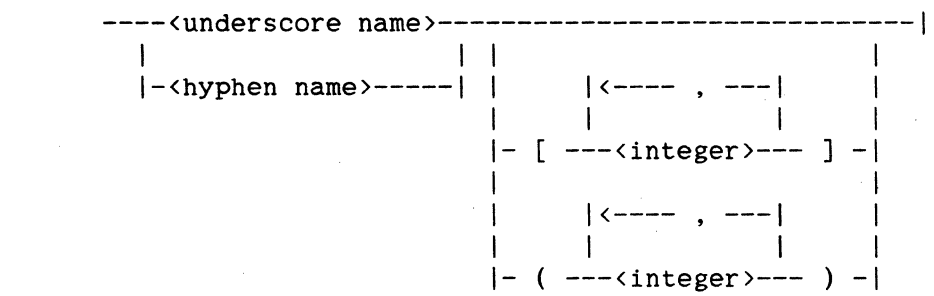
Required Parameters:	Data Type of Parameters:
<structure>	STRING
<item name>	STRING

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

Example

The following program fragments show how ALGOLPUTNULL is used to set a data base item to its NULL value.

Local Declaration of Variables:

```
STRING
      DATA_SET_NAME
      ,ITEM_NAME
      ;
```

Assignment of Values to Parameters:

```
DATA_SET_NAME:="EMP";
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBPUTNULL (STRUCTURE, ITEM);
   STRING STRUCTURE, ITEM;
   LIBRARY DMI (ACTUALNAME = "ALGOLPUTNULL");
```

Invocation of Entry Point:

```
ITEM_NAME:="EMP-SALARY";
RSLT:= DBPUTNULL(DATA_SET_NAME,ITEM_NAME);
IF RSLT
   THEN   EXCEPTIONHANDLER  ;
```

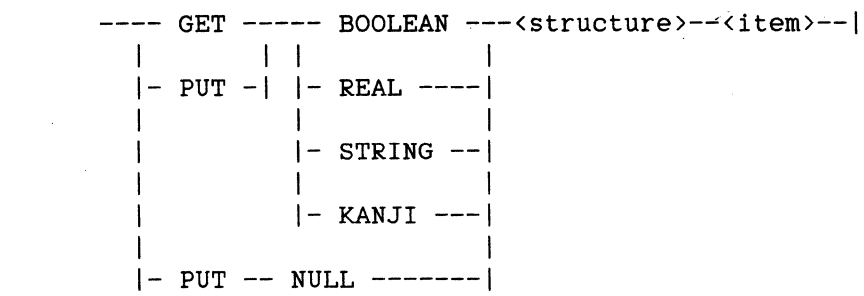
ALGOLDATA

ALGOLDATA allows "GET" and "PUT" commands to be dynamically constructed during program execution. The data request parameter contains a "GET" or "PUT" operation. The data parameter receives the value for "GET" operations and is used to hold the new value for "PUT" operations.

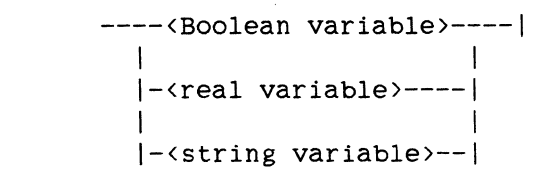
Required Parameters:	Data Type of Parameters:
<data request>	STRING
<data>	STRING

Syntax

<data request>



<data>



Explanation

GET <data type> <structure> <item>

Specifies that a data value of type BOOLEAN, REAL, STRING, or KANJI will be moved from an item in a specified data base structure to a variable declared by the programmer.

PUT <data type> <structure> <item>

Specifies that a data value of type BOOLEAN, REAL, STRING, KANJI, or NULL will be moved from a variable declared by the programmer to an item in a specified data base structure.

<data>

For PUT operations, this parameter holds data that will be moved to the data base. For GET operations, this parameter receives data from the data base.

Example

The following program fragments show how ALGOLDATA is used to move data from an ALGOL program to the data base. The type of operation being used, the type of item being moved, the data set name, the item name, and the data itself are all specified within the body of the program.

Local Declaration of Variables:

```
STRING
      DATA_REQUEST
      ,DATA_BUFFER
      ;
```

Assignment of Values to Parameters:

Values are assigned to the declared variables within the body of the program.

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBDATA (REQUEST, DATA);
   STRING REQUEST, DATA;
   LIBRARY DMI (ACTUALNAME = "ALGOLDATA");
```

Invocation of Entry Point:

```
DATA_REQUEST:= "PUT REAL EMP EMP-DATE-HIRED" ;
DATA_BUFFER:= "811223" ;
RSLT:= DBDATA(DATA_REQUEST,DATA_BUFFER) ;
IF RSLT
   THEN EXCEPTIONHANDLER ;
```

4.3 EXCEPTION-HANDLING ENTRY POINTS

The exception-handling entry points are among the functions of the interpretive interface that are allowed even when the data base is closed.

All ALGOL entry points of the DMINTERPRETER Library except ALGOLEXCEPTIONTEXT and ALGOLEXCEPTIONNAME return a particular value to indicate that an exception has occurred. After an ALGOL entry point call to the DMINTERPRETER Library has been completed, the entry point returns an integer to the Boolean result descriptor in the ALGOL invocation procedure. The returned value is 1 if an exception has occurred and 0 if no exception has occurred. An ALGOL exception is the standard DMSII exception word which includes the exception category and subcategory numbers and the number of the structure involved.

The ALGOLSTATUS entry point can subsequently be called to retrieve the actual exception, which is the same exception as that returned by the last entry point called. The exception result returned by ALGOLSTATUS can be passed to ALGOLEXCEPTIONNAME and/or ALGOLEXCEPTIONTEXT to find out the name of the exception or the full exception message, respectively.

When the ALGOLEXCEPTIONTEXT or ALGOLEXCEPTIONNAME entry point is called, the DMINTERPRETER Library calls the DMSUPPORT Library to retrieve the text or the name of the exception message. The ALGOLEXCEPTIONTEXT or ALGOLEXCEPTIONNAME entry points can return any exception that can be returned by the DMSII user language interface. In addition, the DMSUPPORT Library contains a special category of exceptions called INTLIBERRORS, for errors peculiar to the interpretive interface. (Refer to the appendix "DMSII Exceptions and Errors" for a list of exception categories and subcategories and the corresponding exception messages.)

ALGOLSTATUS

ALGOLSTATUS returns the exception result of the last entry point call that was made. The program does not need to directly examine the contents of of this exception result, but should pass the result on to ALGOLEXCEPTIONNAME or ALGOLEXCEPTIONTEXT, to find out the exception message or exception name from the DMSUPPORT Library.

Required Parameters:	Data Type of Parameters:
None for ALGOL	Not applicable

Example

The following program fragments show how ALGOLSTATUS is used to obtain an exception result from the last entry point that was called.

Local Declaration of Variables:

```
BOOLEAN
    STATUSRSLT
    ;
```

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBSTATUS;
   LIBRARY DMI (ACTUALNAME = "ALGOLSTATUS");
```

Invocation of Entry Point:

```
STATUSRSLT := DBSTATUS;
```

ALGOLEXCEPTIONTEXT

A call on ALGOLEXCEPTIONTEXT should be preceded by a call on ALGOLSTATUS. After ALGOLSTATUS returns the exception-word for the latest call on the DMINTERPRETER Library, ALGOLEXCEPTIONTEXT can return the text of the exception message in its text parameter.

The exception message can later be displayed by the program. The text parameter contains "NO EXCEPTION" if no exception has occurred. If text is not long enough to hold the entire message, the message is truncated to fit in the available space. If the space is longer than the message text, then it is padded with trailing blanks.

If an error message is encountered when ALGOLEXCEPTIONTEXT is called, the user should investigate whether the DMSUPPORT Library is currently unavailable, possibly for security reasons.

Required Parameters:	Data Type of Parameters:
<exception>	BOOLEAN
<text>	STRING

Explanation

<exception>

A BOOLEAN variable containing the exception information from a DBSTATUS entry point call.

<text>

A STRING variable with enough room to hold error messages without truncation if possible.

Example

The following procedure shows how ALGOLEXCEPTIONTEXT is used to obtain the exception message that corresponds to the exception word returned by the ALGOLSTATUS entry point.

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBEXCEPTIONTEXT (EXCEPTION, MSG) ;
   VALUE EXCEPTION;
   BOOLEAN EXCEPTION;
   STRING MSG;
   LIBRARY DMI (ACTUALNAME = "ALGOLEXCEPTIONTEXT");
```

Exception-Handling Procedure:

```
PROCEDURE EXCEPTIONHANDLER;
BEGIN
STRING
    EXCEPTIONMSG
    ;
BOOLEAN
    STATUSRSLT
    ;

    STATUSRSLT:= DBSTATUS ;
    DBEXCEPTIONTEXT(STATUSRSLT,EXCEPTIONMSG);
    WRITE(REMOTEFILE,<"EXCEPTION-TEXT=">);
    WRITE(REMOTEFILE,<A80>,EXCEPTIONMSG);
END  OF EXCEPTIONHANDLER ;
```

ALGOLEXCEPTIONNAME

A call on ALGOLEXCEPTIONNAME should be preceded by a call on ALGOLSTATUS. For an exception returned by ALGOLSTATUS, ALGOLEXCEPTIONNAME returns in the text parameter the name of the type of exception that occurred. Text contains either the name of a DMSII exception category, such as NOTFOUND or OPENERROR, or "NO EXCEPTION" if no exception has occurred. If text is not long enough to hold the entire message, the message is truncated to fit in the available space. If text is longer than the message, the extra space is padded with trailing blanks.

If an error message is encountered when ALGOLEXCEPTIONNAME is called, the user should investigate whether the DMSUPPORT Library is currently unavailable, possibly for security reasons.

Required Parameters:	Data Type of Parameters:
<exception>	BOOLEAN
<text>	STRING

Explanation

<exception>

A BOOLEAN variable containing the exception information from a DBSTATUS entry point call.

<text>

A STRING variable with enough room to hold error messages without truncation if possible.

Example

The following procedure shows how ALGOLEXCEPTIONNAME is used to obtain the name of the exception that corresponds to the exception result returned by the ALGOLSTATUS entry point.

Declaration of Library Entry Point:

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBEXCEPTIONNAME (EXCEPTION, CATEGORY);
   VALUE EXCEPTION;
   BOOLEAN EXCEPTION;
   STRING CATEGORY;
   LIBRARY DMI (ACTUALNAME = "ALGOLEXCEPTIONNAME");
```

Exception-Handling Procedure:

```
PROCEDURE EXCEPTIONHANDLER;
BEGIN
STRING
    EXCEPTIONMSG
    ;
BOOLEAN
    STATUSRSLT
    ;

    STATUSRSLT:= DBSTATUS ;
    DBEXCEPTIONNAME(STATUSRSLT,EXCEPTIONMSG);
    WRITE(REMOTEFILE,<"EXCEPTION-NAME=",
                      A14>,EXCEPTIONMSG);
END  OF EXCEPTIONHANDLER ;
```

4.4 ATTRIBUTE-SETTING ENTRY POINTS

ALGOLSETLIMIT is used to restrict the number of times the DMINTERPRETER can call the Accessroutines to complete a find. FINDLIMIT is the only valid limit type allowed for the ALGOLSETLIMIT entry point.

ALGOLSETLIMIT can be used to declare DMINTERPRETER entry points in application programs, unless renaming is desired or necessary due to limitations of a particular language.

ALGOLSETLIMIT

ALGOLSETLIMIT sets the various limit values for the DMINTERPRETER.

Required Parameters:	Data Type of Parameters:
<limittype>	STRING
<limitvalue>	STRING

Explanation

<limittype>=FINDLIMIT

A string parameter that specifies the type of limit to be set, and specifies the maximum number of times the DMINTERPRETER can call the Accessroutines to perform a FIND or LOCK.

FINDLIMIT is the only <limittype> currently defined. By setting FINDLIMIT the user can prevent long linear searches.

If an invalid <limittype> is specified, INTLIBERROR 33 is returned. If the DMINTERPRETER exceeds the FINDLIMIT while performing a FIND or LOCK, INTLIBERROR 35 is returned.

<limitvalue>

An integer string parameter which specifies the value to be assigned to <limittype>. Negative values are not allowed for <limittype>. Specifying a <limitvalue> of zero disables the FINDLIMIT mechanism, allowing infinitely long linear searches.

When the DMINTERPRETER is first initialized, FINDLIMIT is set to the default value of zero.

5 ENTRY POINTS IN PL/I

This section explains the PL/I interface to the DMINTERPRETER Library, and provides a sample application program written in PL/I to illustrate the following concepts:

1. How entry points of the DMINTERPRETER Library are declared in a PL/I application program.

2. How a PL/I application program "calls" or "invokes" entry points of the DMINTERPRETER Library.

3. How the DMINTERPRETER Library returns results to PL/I programs.

4. How exceptions returned by entry point calls are handled.

This section also lists all entry points of the DMINTERPRETER Library for PL/I and describes their functions and arguments. The following information is included under each description of an entry point:

1. A brief explanation of the function of the entry point.

2. Syntax diagrams and semantics that describe required arguments. (More detailed syntax diagrams for data base constructs used with the interpretive interface are given in the appendix "Syntax Diagrams for Entry Point Parameters.")

3. A program fragment that illustrates local declaration of the entry point in an application program.

4. A program fragment that illustrates the assignment of values to variables used as arguments.

5. A program fragment that illustrates the invocation of the entry point.

Table 5-1 lists all of the ALGOL entry points, which PL/I programs call directly. (Separate entry points for PL/I only are not needed, because word sizes and data types for ALGOL and PL/I are very similar.) The entry points below are listed in the order in which they are described later in this section.

Table 5-1. Summary
of PL/I Entry Points

Standard Entry Points:

ALGOLOPEN
ALGOLCLOSE
ALGOLBEGINTRANSACTION
ALGOLENDTRANSACTION
ALGOLCREATE
ALGOLRECREATE
ALGOLSTORE
ALGOLFREE
ALGOLSET
ALGOLFIND
ALGOLLOCK
ALGOLDELETE
ALGOLVERB

Data Transfer Entry Points:

ALGOLGETSTRING
ALGOLGETKANJI
ALGOLGETREAL
ALGOLGETDOUBLE
ALGOLGETBOOLEAN
ALGOLPUTSTRING
ALGOLPUTKANJI
ALGOLPUTREAL
ALGOLPUTDOUBLE
ALGOLPUTBOOLEAN
ALGOLPUTNULL
ALGOLDATA

Exception Handling Entry Points:

ALGOLSTATUS
ALGOLEXCEPTIONTEXT
ALGOLEXCEPTIONNAME

Attribute Setting Entry Point:

ALGOLSETLIMIT

LIBRARY DECLARATION

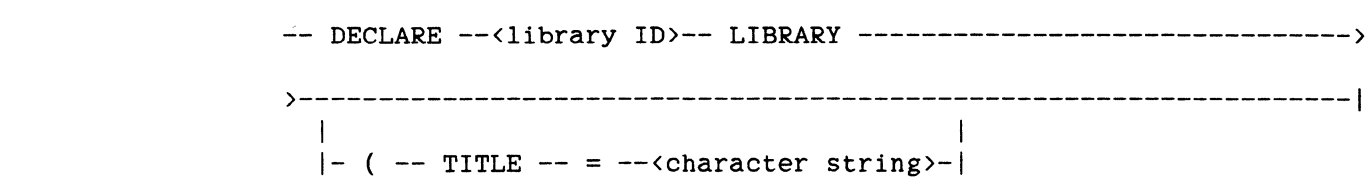
The first step in writing an application program that accesses the interpretive interface is to declare the DMINTERPRETER Library and its entry points within the application program. Linkage to the DMINTERPRETER Library must be direct, because the library actually contains the procedures that are named in its EXPORT declarations.

In PL/I, linkage to the DMINTERPRETER Library is accomplished by declaring the library, and then assigning the external name of the library code file (which has been compiled for a particular data base) to the library identifier by using the TITLE attribute. The library identifier is used as an internal name within the application program to refer to the DMINTERPRETER Library.

The following diagrams show explicit library declaration in PL/I, using the LIBRARY attribute in a DECLARE statement:

Syntax

<library declaration>



<library ID>



Explanation

The library declaration declares an identifier to be a library ID, and specifies a value for the TITLE attribute associated with the library. A PL/I identifier is a string of alphanumeric and break characters that begins with an alphabetic character, having a maximum length of 63 characters. An identifier is not part of a comment or string constant and is enclosed by delimiters.

In the TITLE clause, character string is a constant of zero or more characters in the data character set enclosed in single quotation marks ('). (For more information on PL/I basic constructs, refer to the PL/I Reference Manual.) If the TITLE clause is used in the library declaration, linkage to the DMINTERPRETER Library is complete.

If the short form of library declaration (that is, "DECLARE <library ID> LIBRARY") is used, the title of the library code file which has been compiled for a particular data base must be specified in the following manner:

```
TITLE (<library ID>) = "character string"
```

In the sample PL/I program provided in the appendix "Sample Programs," the short form of library declaration has been used. The library ID is DMI, and the title of the library code file is specified as follows:

```
TITLE(DMI) = DMI_NAME ;
```

DMI_NAME is a variable specified in the Local Declarations block of the program as follows:

```
DMI_NAME        CHAR(24) INIT('DMINTERPRETER/EMPJOB')
```

Libraries may be declared in any block of a user program. The library and its entry points are valid within the scope of the block; upon exiting the block, linkage to the library is broken and the count of users is decremented.

ENTRY-POINT DECLARATION

To ease the process of writing programs that access the DMINTERPRETER Library, the DATABASE/DMINTERPRETER symbolic file provided on the release tapes contains a declaration of the library and its entry points for each interface language. When using interface languages other than PL/I, this declaration can be included in application programs by using the $ INCLUDE compiler control option for the appropriate sequence range within DATABASE/DMINTERPRETER. However, the $ INCLUDE compiler control option is not available to PL/I, so the user must insert the desired PL/I entry-point declarations using Command AND Edit (CANDE). The correct sequence range within DATABASE/DMINTERPRETER is

20300000 - 20341000

The entry point declarations rename exported ALGOL entry points as follows:

Exported Name of Entry Point:	Renamed As:
ALGOLOPEN	DBOPEN
ALGOLCLOSE	DBCLOSE
ALGOLBEGINTRANSACTION	DBBEGINTRANSACTION
ALGOLENDTRANSACTION	DBENDTRANSACTION
ALGOLCREATE	DBCREATE
ALGOLRECREATE	DBRECREATE
ALGOLSTORE	DBSTORE
ALGOLFREE	DBFREE
ALGOLSET	DBSET
ALGOLFIND	DBFIND
ALGOLLOCK	DBLOCK
ALGOLDELETE	DBDELETE
ALGOLVERB	DBVERB
ALGOLGETSTRING	DBGETCHARACTER
ALGOLGETKANJI	DBGETKANJI
ALGOLGETREAL	DBGETFLOAT
ALGOLGETDOUBLE	DBGETDOUBLE
ALGOLGETBOOLEAN	DBGETBIT
ALGOLPUTSTRING	DBPUTCHARACTER
ALGOLPUTKANJI	DBPUTKANJI
ALGOLPUTREAL	DBPUTFLOAT
ALGOLPUTDOUBLE	DBPUTDOUBLE
ALGOLPUTBOOLEAN	DBPUTBIT
ALGOLPUTNULL	DBPUTNULL
ALGOLDATA	DBDATA
ALGOLSTATUS	DBSTATUS
ALGOLEXCEPTIONTEXT	DBEXCEPTIONTEXT
ALGOLEXCEPTIONNAME	DBEXCEPTIONNAME

Data types for the arguments of each entry point are given under individual entry point descriptions later in this section. The sequence range from DATABASE/DMINTERPRETER that can be inserted using CANDE declares all entry points, renaming them with the "DB" prefix. Users who want to rename the exported ALGOL entry points in a different manner, or declare only a subset of the entry points, must individually declare the entry points within the application program.

The following diagram shows how a library entry point is declared in PL/I:

Syntax

<library entry point declaration>

```
    -- DECLARE --<entry point name>-- ENTRY ---------------------------->

    >------------------------------ RETURNS -- (BIT(48)) --------------|
      |                             |
      |- (<argument data type>) -|
```

<entry point name>

```
    --<identifier>--|
```

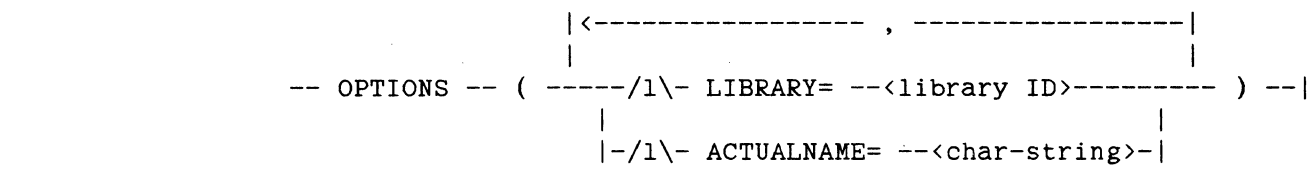
Explanation

The library entry point declaration declares a procedure to be an entry point in the library known to this program through the library ID. The procedure cannot be declared FORWARD or EXTERNAL.

Association of an entry point name with a library is accomplished by adding the options list to the library entry point declaration. Syntax for the options list is as follows:

Syntax

<entry point options list>



Explanation

The library must be named; the ACTUALNAME is optional, the declared identifier being used by default. The ACTUALNAME clause allows the renaming of a library entry point by using a different identifier in the library entry point declaration which can still link to the appropriate entry point through ACTUALNAME. The character string in the "ACTUALNAME=" clause cannot contain any leading, trailing, or embedded blanks and must be a valid PL/I identifier. The char-string is the actual name that is given for this procedure in the template of this library.

Example

The following example shows how an entry point may be declared within a PL/I program:

```
DECLARE DBOPEN ENTRY (CHARACTER(*))
     RETURNS (BIT(48)
     OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLOPEN');
```

INVOCATION OF ENTRY POINTS

In PL/I, each entry point of the DMINTERPRETER Library is a procedure that returns a bit string containing 48 bits. The following is a typical method of invoking an entry point of the DMINTERPRETER Library, where RSLT is a 48-bit bit string, and EXCEPTIONHANDLER is a procedure for handling exceptions:

How to Invoke an Entry Point:

```
RSLT ^= DBOPEN(OPEN_TYPE) ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER();
```

After an entry point call on the DMINTERPRETER Library, a result descriptor is returned to the program. This result descriptor is a 48-bit word which is the standard Data Management System II (DMSII) exception word; it includes the exception category and subcategory, and the structure number involved. When a call on the DMINTERPRETER Library causes an exception, the last bit in the 48-bit word is set to 1. (When a call on the library is successful, the 48-bit word contains all zeros.) For more information on exceptions, refer to "Exception-Handling Entry Points" later in this section.

5.1 STANDARD ENTRY POINTS

Standard entry points correspond to functions performed by DMSII user language interface statements. The results returned by standard entry points are used for exception handling.

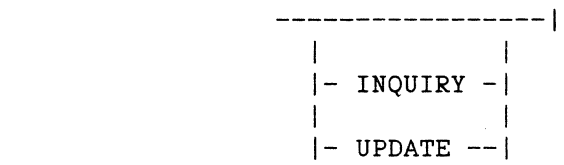
ALGOLOPEN

ALGOLOPEN opens the data base for inquiry or update, depending on which open disposition is used. ALGOLOPEN must be called before any data base operations are performed. ALGOLOPEN, the exception handling entry points (ALGOLSTATUS, ALGOLEXCEPTIONTEXT, and ALGOLEXCEPTIONNAME), and operations on the intrinsic data set DBSTRUCTURE are the only functions allowed when the data base is closed.

Required Arguments:	Data Types of Arguments:
<open disposition>	CHARACTER

Syntax

<open disposition>



Explanation

INQUIRY

Causes the data base to be opened for INQUIRY. No updates are allowed.

UPDATE

Causes the data base to be opened for UPDATE. Both inquiry and update are allowed.

Example

The following program fragments show how ALGOLOPEN is used to open the data base for UPDATE. The OPEN_TYPE variable must be set to UPDATE or INQUIRY before invoking the entry point.

Setting Variable to Initial Values:

```
DCL
     OPEN_TYPE              CHAR( 6) INIT('UPDATE')
     ;
```

Library Declaration of Entry Point:

```
DECLARE DMI LIBRARY;
DECLARE DBOPEN ENTRY (CHARACTER(*))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLOPEN');
```

Invocation of Entry Point:

```
RSLT:= DBOPEN(OPEN_TYPE) ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER();
```

ALGOLCLOSE

ALGOLCLOSE explicitly closes the data base. (The data base is implicitly closed when the library execution completes.)

Required Arguments:	Data Types of Arguments:
None	Not applicable

Example

The following example shows how ALGOLCLOSE is used to close the data base. When invoking ALGOLCLOSE, no arguments need to be specified, because none are required for this entry point.

Setting Variables to Initial Values:

Not applicable.

Library Declaration of Entry Point:

```
DECLARE DBCLOSE ENTRY
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLCLOSE');
```

Invocation of Entry Point:

```
RSLT:= DBCLOSE ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER();
```

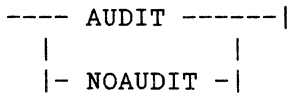
ALGOLBEGINTRANSACTION

ALGOLBEGINTRANSACTION causes the user program to enter transaction state. The audit argument specifies whether an image of the current restart data set record is to be audited. Calls on ALGOLBEGINTRANSACTION are ignored for unaudited data bases.

Required Arguments:	Data Types of Arguments:
<audit>	CHARACTER

Syntax

<audit>



Explanation

AUDIT

Specifies that an image of the current restart data set will be audited.

NOAUDIT

Specifies that auditing an image of the current restart data set is not required. NOAUDIT is the default value of the audit argument.

Example

The following example shows how a user program can enter transaction state by calling ALGOLBEGINTRANSACTION. The audit argument must be set to AUDIT or NOAUDIT before invoking the entry point.

Setting Variables to Initial Values:

```
DCL
      P_AUDIT                CHAR( 5) INIT('AUDIT')
      ;
```

Library Declaration of Entry Point:

```
DECLARE DBBEGINTRANSACTION ENTRY (CHARACTER(*))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI,
            ACTUALNAME = 'ALGOLBEGINTRANSACTION');
```

Invocation of Entry Point:

```
RSLT:= DBBEGINTRANSACTION(P-AUDIT) ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER();
```

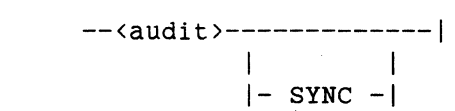
ALGOLENDTRANSACTION

ALGOLENDTRANSACTION causes the user program to leave transaction state. ALGOLENDTRANSACTION should be used only with audited data bases. Calls on ALGOLENDTRANSACTION are ignored for unaudited data bases.

Required Arguments:	Data Types of Arguments:
<auditsync>	CHARACTER

Syntax

<auditsync>



Explanation

<audit>

Specifies whether an image of the current restart data set will be audited (this is the audit argument discussed under ALGOLBEGINTRANSACTION in this section).

<audit> SYNC

Forces a syncpoint, when an image of the current restart data set is audited.

Example

The following program fragments show how ALGOLENDTRANSACTION is used to cause a program to exit transaction state, with the current restart data set record being audited. The P_AUDIT variable must be set to AUDIT or AUDIT SYNC before invoking the entry point.

Setting Variables to Initial Values:

```
DCL
       P_AUDIT                  CHAR( 5) INIT('AUDIT')
       ;
```

Library Declaration of Entry Point:

```
DECLARE DBENDTRANSACTION ENTRY (CHARACTER(*))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI,
            ACTUALNAME = 'ALGOLENDTRANSACTION');
```

Invocation of Entry Point:

```
RSLT:= DBENDTRANSACTION(P_AUDIT) ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER();
```

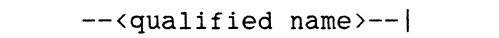
ALGOLCREATE

ALGOLCREATE creates a new record for the specified data set. The record type argument specifies whether the record will be fixed-length or variable-format.

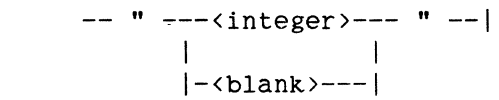
Required Arguments:	Data Types of Arguments:
<data set>	CHARACTER
<record type>	CHARACTER

Syntax

<data set>



<record type>



Explanation

<data set>

Names a data set for which a new record will be created.

"<integer>"

A non-zero integer specifies a variable-length record. If the specified integer is 0, the fixed part of a variable-length record is created.

"<blank>"

If an empty string is specified, a fixed-length record will be created.

Example

The following program fragments show how ALGOLCREATE is used to create a new fixed-length record belonging to a data set named EMP.

Setting Variables to Initial Values:

```
DCL
     DATA_SET_NAME          CHAR(17) INIT('EMP'),
     SPACE_1                  CHAR( 1) INIT(' ')
     ;
```

Library Declaration of Entry Point:

```
DECLARE DBCREATE ENTRY (CHARACTER(*),CHARACTER(*))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLCREATE');
```

Invocation of Entry Point:

```
RSLT:= DBCREATE(DATA_SET_NAME,SPACE_1) ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER();
```

ALGOLRECREATE

ALGOLRECREATE re-creates, or copies, a new image of an existing record for the purpose of modification. The data set argument names the existing data set where the record resides. The record type argument specifies whether the re-created record will be fixed-length or variable-format.

Required Arguments:	Data Types of Arguments:
<data set>	CHARACTER
<record type>	CHARACTER

Syntax

<data set>

```
--<qualified name>--|
```

<record type>

```
-- " ---<integer>--- " --|
      |           |
      |-<blank>---|
```

Explanation

<data set>

Names a data set for which a record will be re-created.

"<integer>"

A non-zero integer specifies a variable-length record. If the specified integer is 0 (zero), the fixed part of a variable-length record is created.

"<blank>"

If an empty string is specified, a fixed-length record will be created.

Example

The following program fragments show how ALGOLRECREATE is used to re-create an image of an existing fixed-length record belonging to a data set named EMP.

Setting Variables to Initial Values:

```
DCL
      DATA_SET_NAME          CHAR(17) INIT('EMP'),
      SPACE_1                  CHAR(1)  INIT (' ')
      ;
```

Library Declaration of Entry Point:

```
DECLARE DBRECREATE ENTRY (CHARACTER(*))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI,
                  ACTUALNAME = 'ALGOLRECREATE');
```

Invocation of Entry Point:

```
RSLT:= DBRECREATE(DATA_SET_NAME,SPACE_1) ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER();
```

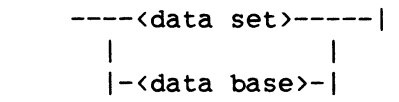
ALGOLSTORE

ALGOLSTORE stores a record in the specified structure. For audited data bases, ALGOLBEGINTRANSACTION must have been called before ALGOLSTORE is called, because the program must be in transaction state in order for a record to be stored. If ALGOLSTORE is called outside of transaction state for an audited data base, an exception is returned.

Required Arguments:	Data Types of Arguments:
<structure>	CHARACTER

Syntax

<structure>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

Example

The following program fragments show how ALGOLSTORE is used to store a record in a data set named EMP.

Setting Variables to Initial Values:

```
DCL
      DATA_SET_NAME       CHAR(17) INIT('EMP')
      ;
```

Library Declaration of Entry Point:

```
DECLARE DBSTORE ENTRY (CHARACTER(*))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLSTORE');
```

Invocation of Entry Point:

```
RSLT:= DBSTORE(DATA_SET_NAME) ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER();
```

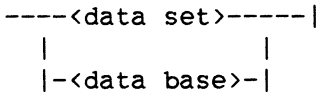
ALGOLFREE

ALGOLFREE unlocks a data set record or a global data record. ALGOLFREE may be used to unlock a record that you anticipate may not be implicitly freed for a long time. After ALGOLFREE is called for a record, other programs may lock that record.

Required Arguments:	Data Types of Arguments:
<structure>	CHARACTER

Syntax

<structure>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

Example

The following program fragments show how ALGOLFREE is used to unlock a record belonging to a specified structure. In this case, a record in the data set EMP is being unlocked.

Setting Variables to Initial Values:

```
DCL
      DATA_SET_NAME       CHAR(17) INIT('EMP')
      ;
```

Library Declaration of Entry Point:

```
DECLARE DBFREE ENTRY (CHARACTER(*))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLFREE');
```

Invocation of Entry Point:

```
RSLT:= DBFREE(DATA_SET_NAME) ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER();
```

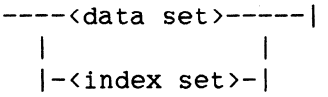
ALGOLSET

ALGOLSET sets particular data sets or index sets to their beginning or ending points, or sets the position pointer of an index set to the current record of a particular data set. The following syntax diagrams and semantics explain which structures can be set to which targets.

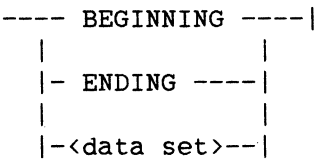
Required Arguments:	Data Types of Arguments:
<structure>	CHARACTER
<target>	CHARACTER

Syntax

<structure>



<target>



Explanation

<data set> BEGINNING

Names a particular data set that will be set to its beginning point at run time.

<data set> ENDING

Names a particular data set that will be set to its ending point at run time.

<index set> <data set>

Names the data set spanned by the index set, and sets the position pointer of the index set to the current record within the spanned data set.

<index set> BEGINNING

Names a particular index set that will be set to its beginning point at run time.

<index set> ENDING

Names a particular index set that will be set to its ending point at run time.

Example

The following program fragments show how ALGOLSET is used to set a particular data base structure to a desired target. In this case, the data set named EMP is set to its beginning point.

Setting Variables to Initial Values:

```
DCL
      DATA_SET_NAME       CHAR(17) INIT('EMP'),
      BEGIN_1             CHAR( 9) INIT('BEGINNING')
      ;
```

Library Declaration of Entry Point:

```
DECLARE DBSET ENTRY (CHARACTER(*), CHARACTER(*))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLSET');
```

Invocation of Entry Point:

```
RSLT:= DBSET(DATA_SET_NAME, BEGIN_1) ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER();
```

ALGOLFIND

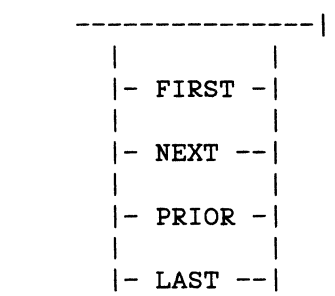
ALGOLFIND transfers a record to the user work area associated with the specified data base structure. The other arguments shown below indicate the direction in which the FIND operation proceeds, and the selection expression used during the search.

An ALGOLFIND call is more sophisticated than a user language interface FIND statement. Several restrictions have been eased, and some algorithmic enhancements have been made. These improvements are discussed in detail in the appendix "DBFIND Enhancements."

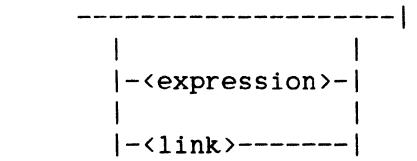
Required Arguments:	Data Types of Arguments:
<direction>	CHARACTER
<structure>	CHARACTER
<condition>	CHARACTER

Syntax

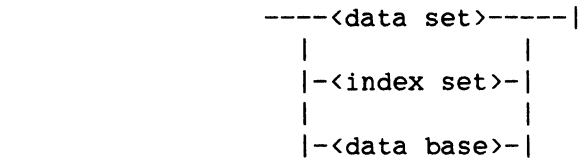
<direction>



<condition>



<structure>



Explanation

(empty)

If the direction argument is empty, then the ALGOLFIND operation will transfer the current record in the specified structure to the user work area.

FIRST

Indicates that the ALGOLFIND operation will transfer the first record in the specified structure to the user work area.

NEXT

Indicates that the ALGOLFIND operation will transfer the next record in the specified structure to the user work area.

PRIOR

Indicates that the ALGOLFIND operation will transfer the prior record in the specified structure to the user work area.

LAST

Indicates that the ALGOLFIND operation will transfer the last record in the specified structure to the user work area.

<data set>

Names a data set in the data base that will be accessed at run time.

<index set>

Names an index set in the data base that will be used for the search.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<expression>

A selection expression used during the ALGOLFIND operation to find records that meet a specified condition.

<link>

Specifies the data base structure or data set item that links the current record to a record in another data set.

Example

The following program fragments show how ALGOLFIND is used to find the first record in a data set named EMP, which satisfies a specified condition.

Setting Variables to Initial Values:

```
DCL
      FIRST_1                CHAR( 5) INIT('FIRST'),
      DATA_SET_NAME          CHAR(17) INIT('EMP'),
      CONDITION_1            CHAR( 9) INIT('EMP-NO=11')
      ;
```

Library Declaration of Entry Point:

```
DECLARE DBFIND ENTRY (CHARACTER(*),
                      CHARACTER(*),CHARACTER(*))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLFREE');
```

Invocation of Entry Point:

```
RSLT:= DBFIND(FIRST_1,DATA_SET_NAME,CONDITION_1);
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER();
```

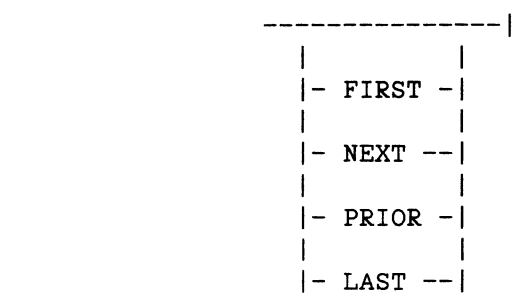
ALGOLLOCK

ALGOLLOCK functions in the same manner as ALGOLFIND, except that a record is locked after it is found. ALGOLLOCK transfers a record to the user work area associated with the specified data structure and locks it there for modification. The other arguments shown below indicate the direction in which the search for the record proceeds, and the selection expression used during the search.

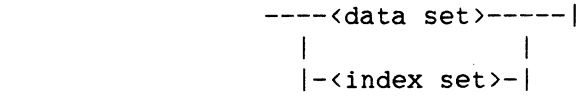
Required Arguments:	Data Types of Arguments:
<direction>	CHARACTER
<structure>	CHARACTER
<condition>	CHARACTER

Syntax

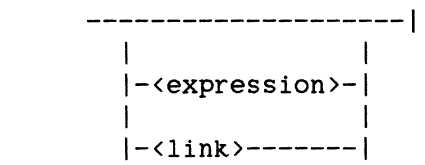
<direction>



<structure>



<condition>



Explanation

(empty)

If the direction argument is empty, the default value is used, and the ALGOLLOCK operation transfers the current record in the specified structure to the user work area.

FIRST

Indicates that the ALGOLLOCK operation will transfer the first record in the specified structure to the user work area.

NEXT

Indicates that the ALGOLLOCK operation will transfer the next record in the specified structure to the user work area.

PRIOR

Indicates that the ALGOLLOCK operation will transfer the prior record in the specified structure to the user work area.

LAST

Indicates that the ALGOLLOCK operation will transfer the last record in the specified structure to the user work area.

<data set>

Names a data set in the data base that will be accessed at run time.

<index set>

Names an index set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<expression>

A selection expression used during the ALGOLLOCK operation to find and lock records that meet a specified condition.

<link>

Specifies the data base structure or data set item that links the current record to a record in another data set.

Example

The following program fragments show how ALGOLLOCK is used to find and lock the first record in a data set named EMP, which satisfies a specified condition.

Setting Variables to Initial Values:

```
DCL
      FIRST_1                CHAR( 5) INIT('FIRST'),
      DATA_SET_NAME          CHAR(17) INIT('EMP'),
      CONDITION_1            CHAR( 9) INIT('EMP-NO=11')
      ;
```

Library Declaration of Entry Point:

```
DECLARE DBLOCK ENTRY (CHARACTER(*),
                      CHARACTER(*),CHARACTER(*))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLLOCK');
```

Invocation of Entry Point:

```
RSLT:= DBLOCK(FIRST_1,DATA_SET_NAME,CONDITION_1);
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER();
```

ALGOLDELETE

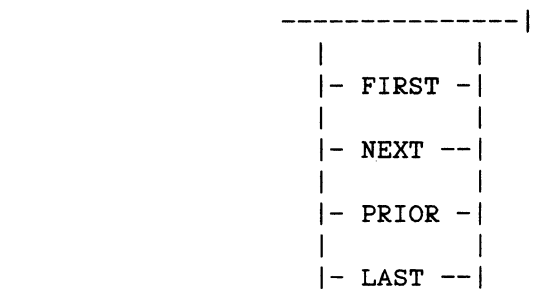
ALGOLDELETE deletes a record from the specified data set. The arguments shown below indicate the direction in which the search and deletion operation proceeds, and the selection expression used during the search.

ALGOLDELETE functions in the same manner as ALGOLFIND, except that a record is deleted after it is found, unless it is a global data record which cannot be deleted. Whenever ALGOLDELETE is called, the data set record is implicitly locked before it is deleted.

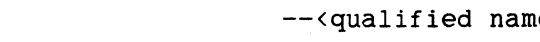
Required Arguments:	Data Types of Arguments:
<direction>	CHARACTER
<data set>	CHARACTER
<condition>	CHARACTER

Syntax

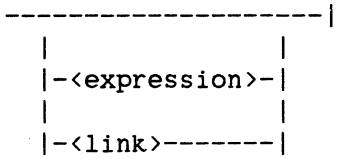
<direction>



<data set>



<condition>



Explanation

(empty)

If the <direction> argument is empty, then the ALGOLDELETE call will transfer the current record in the specified structure to the user work area before deleting the record.

FIRST

Indicates that the ALGOLDELETE call will transfer the first record in the specified structure to the user work area before deleting the record.

NEXT

Indicates that the ALGOLDELETE call will transfer the next record in the specified structure to the user work area before deleting the record.

PRIOR

Indicates that the ALGOLDELETE call will transfer the prior record in the specified structure to the user work area before deleting the record.

LAST

Indicates that the ALGOLDELETE call will transfer the last record in the specified structure to the user work area before deleting the record.

<qualified name>

See the appendix "Syntax Diagrams for Entry Point Parameters" for basic data base constructs that explain the meaning of qualified name in detail.

<expression>

A selection expression used during the ALGOLDELETE operation to find and delete records that meet a specified condition.

<link>

Specifies the data base structure or data set item that links the current record to a record in another data set.

Example

The following program fragments show how ALGOLDELETE is used to find and delete the first record in a data set named EMP, which satisfies a specified condition.

Setting Variables to Initial Values:

```
DCL
      FIRST_1                CHAR( 5) INIT('FIRST'),
      DATA_SET_NAME          CHAR(17) INIT('EMP'),
      CONDITION_1            CHAR( 9) INIT('EMP-NO=11')
      ;
```

Library Declaration of Entry Point:

```
DECLARE DBDELETE ENTRY (CHARACTER(*),
                        CHARACTER(*),CHARACTER(*))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLDELETE');
```

Invocation of Entry Point:

```
RSLT:= DBDELETE(FIRST_1,DATA_SET_NAME,CONDITION_1) ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER();
```

ALGOLVERB

ALGOLVERB causes a specified user language interface statement to be performed. The arguments that are used must evaluate to a complete statement. Permissible DMSII user language constructs are shown in the syntax diagrams below, but exception actions and input/output mapping clauses are not allowed. Brief descriptions of these statements are given below, but full descriptions are given in the DMSII User Language Interface Programming Guide.

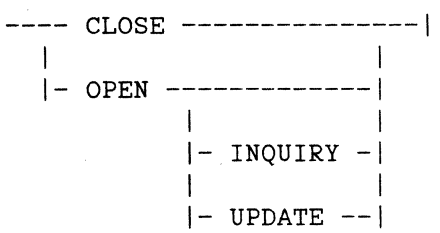
ALGOLVERB allows flexible, run-time definition of DMSII operations, and provides users with an easy transition from DMSII user language interface statements to DMINTERPRETER statements. But because it must completely parse its input, ALGOLVERB runs at higher processor cost than the other, more specialized entry points.

Required Arguments:
-------- ----------

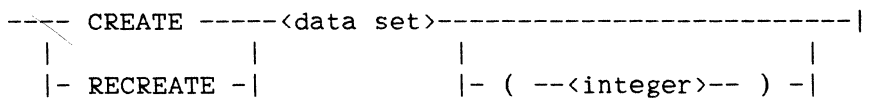
<user language interface statement>

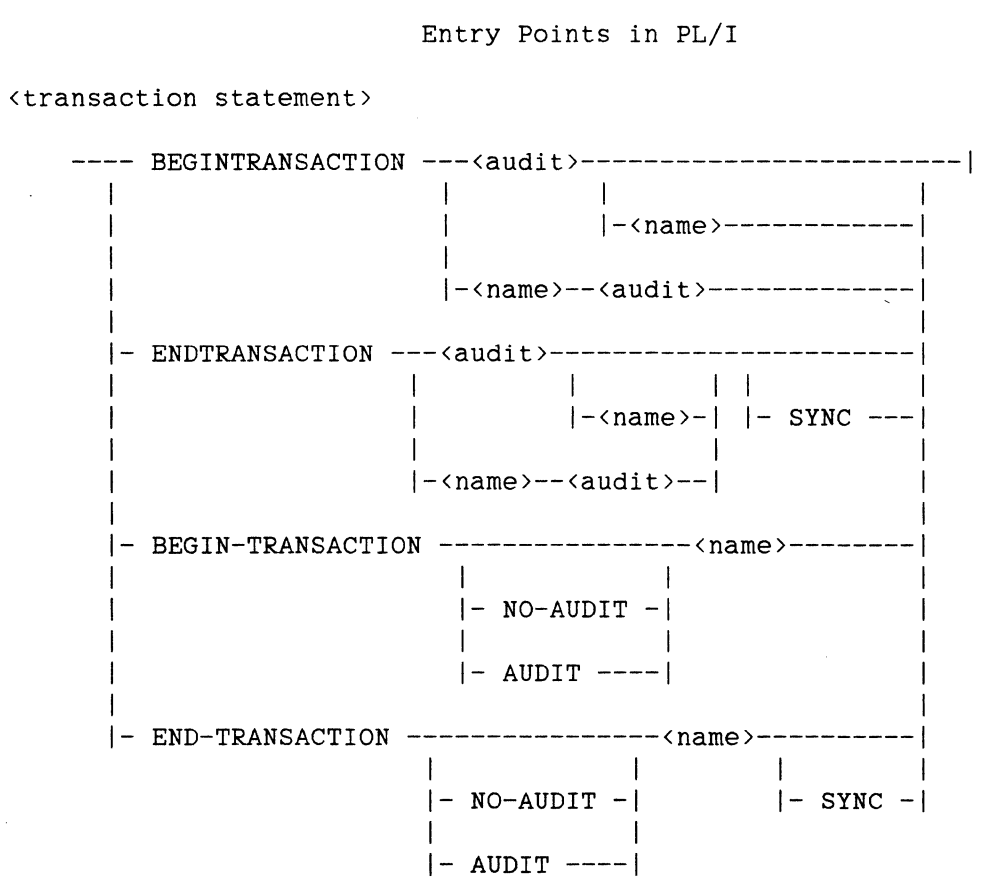
Syntax

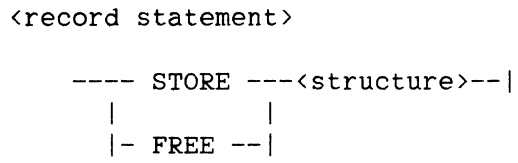
<data base statement>

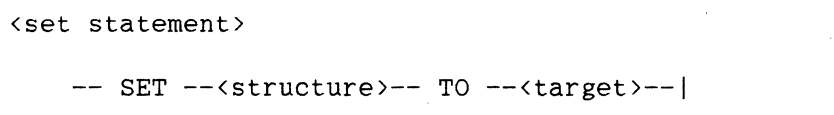


<create statement>

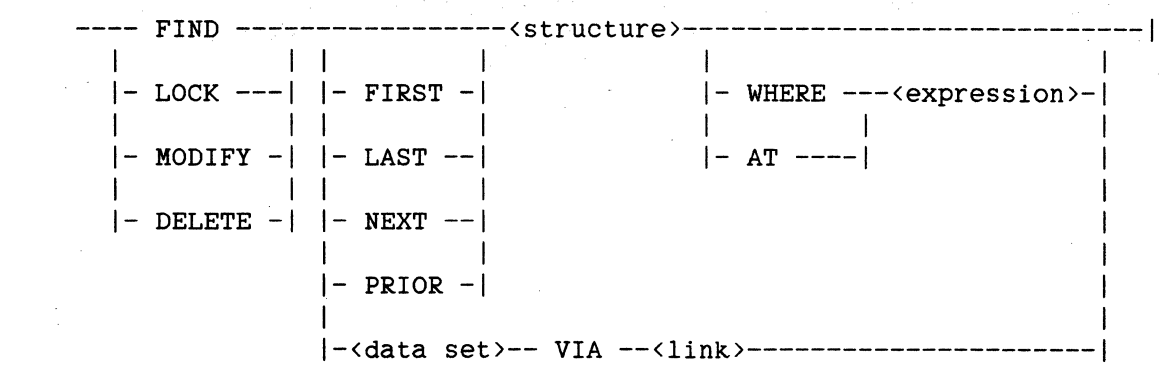



<transaction statement>
BEGINTRANSACTION
<audit>
<name>
<name>
<audit>
ENDTRANSACTION
<audit>
<name>
SYNC
<name>
<audit>
BEGIN-TRANSACTION
<name>
NO-AUDIT
AUDIT
END-TRANSACTION
<name>
NO-AUDIT
SYNC
AUDIT


<record statement>
STORE
<structure>
FREE


<set statement>
SET
<structure>
TO
<target>

<search statement>



<status statement>



Explanation

<data base statement>

Closes the data base, or opens the data base for inquiry or update.

<create statement>

Initializes the user work area for an existing data set record or a re-created image of a data set record.

<transaction statement>

Causes a program to enter transaction state or exit transaction state, with or without auditing an image of the current restart data set and forcing a syncpoint. The name specifies the restart data set, which is checked for validity when it is present.

<record statement>

Stores a new or modified record in a specified structure, or unlocks a record residing in a specified structure.

<set statement>

Sets a specified structure to its beginning or ending, or to a particular data set if the structure is an index set.

<search statement>

Finds a selected record in a specified structure, and locks the record for modification or deletion.

<status statement>

Allows the user to find out whether a prior data base operation was successful.

Example

The following program fragments show how ALGOLVERB can be called to execute a user language interface statement. When using this entry point, arguments for which values have been initialized in the program are not used. Instead, complete user language interface statements must be used. However, when using the <status statement> construct, just the string "DMSTATUS" is passed as an argument.

Setting Variables to Initial Values:

Not applicable when using DBVERB.

Library Declaration of Entry Point:

```
DECLARE DBVERB ENTRY (CHARACTER(*))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLVERB');
```

Invocation of Entry Point:

```
/*                                                          */
/*          USING DBVERB ON A SET STATEMENT                 */
/*                                                          */
      RSLT = DBVERB('SET EMP TO BEGINNING') ;
      IF RSLT ^= 0
         THEN CALL EXCEPTIONHANDLER() ;
/*                                                          */
/*          USING DBVERB ON A SEARCH STATEMENT              */
/*                                                          */
      RSLT = DBVERB('FIND FIRST EMP AT EMP-NO=11') ;
      IF RSLT ^= 0
         THEN CALL EXCEPTIONHANDLER() ;
/*                                                          */
/*           USING DBVERB ON A STATUS STATEMENT             */
/*                                                          */
      RSLT = DBVERB('FIND PRIOR EMP') ;
      RSLT = DBVERB('DMSTATUS') ; /* DOES NOT UPDATE RSLT */
      CALL DBEXCEPTIONTEXT(RSLT,EXCEPTIONMSG) ;
```

5.2 DATA TRANSFER ENTRY POINTS

Data transfer entry points, prefixed with "GET" or "PUT," are provided for moving data to and from user work areas associated with specified data base structures. Each item needed from a data set record must be individually moved to or from an PL/I variable, by calling an appropriate entry point of the DMINTERPRETER Library, before the item can be manipulated by the program.

The "GET" entry points move data from the data base to variables that have been declared by the programmer. The "PUT" entry points move data from PL/I expressions to the data base.

The DMINTERPRETER Library checks the arguments of data transfer entry points to ensure that the requested operations are compatible with the Data and Structure Definition Language (DASDL) data base description. For example, items declared in the data base description with an OCCURS clause must be subscripted.

The "PUT" entry points (ALGOLPUTDISPLAY, ALGOLPUTREAL, ALGOLPUTBOOLEAN, ALGOLPUTDOUBLE, and ALGOLPUTNULL) return an exception if the value to be put into the data base does not fit into the specified data item. The "GET" entry points (ALGOLGETDISPLAY, ALGOLGETREAL, ALGOLGETDOUBLE, and ALGOLGETBOOLEAN) return an exception if the declared variable is too short to hold the specified data item.

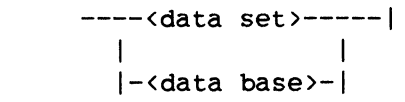
ALGOLGETSTRING

ALGOLGETSTRING sets a variable declared in the application program to a character string representing the contents of an ALPHA item or GROUP item from the current record in the specified structure.

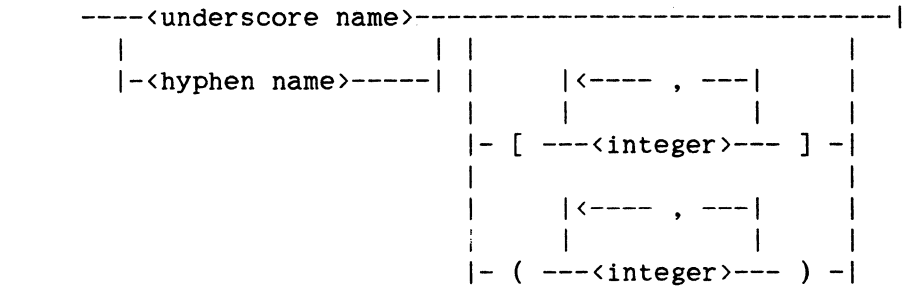
Required Arguments:	Data Types of Arguments:
<structure>	CHARACTER
<item name>	CHARACTER
<variable>	CHARACTER VARYING

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

A PL/I character string variable of varying length.

Example

The following program fragments show how ALGOLGETSTRING is used to move a data item value from the user work area to a variable declared by the programmer. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the value for the ITEM_NAME argument is assigned in the body of the program.

Setting Variables to Initial Values:

```
DCL
       DATA_SET_NAME          CHAR(17) INIT('EMP'),
       STRING_EXPR              CHAR(10) VAR
       ;
```

Library Declaration of Entry Point:

```
DECLARE DBGETCHARACTER ENTRY (CHARACTER(*),
                              CHARACTER(*),
                              CHARACTER(*) VARYING)
RETURNS (BIT(48))
OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLGETSTRING');
```

Invocation of Entry Point:

```
ITEM_NAME = 'EMP-FNAME' ;
RSLT = DBGETCHARACTER(DATA_SET_NAME,
                      ITEM_NAME,STRING_EXPR) ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER() ;
```

ALGOLGETKANJI

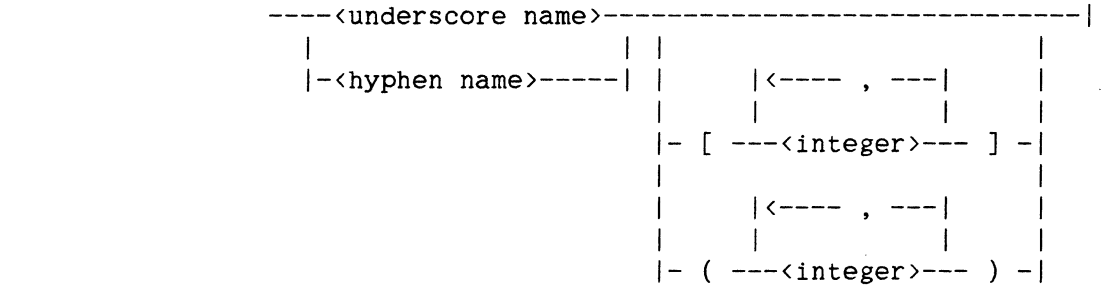
ALGOLGETKANJI sets a variable declared in the application program to a character string representing the contents of an ALPHA item of which USAGE IS KANJI from the current record in the specified structure.

Required Parameters:	Data Type of Parameters:
<structure>	CHARACTER
<item name>	CHARACTER
<variable>	CHARACTER VARYING

Syntax

<structure>

<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

A PL/I character string variable of varying length.

Example

The following program fragments show how ALGOLGETKANJI is used to move a data item value from the DMSII user work area to a variable declared by the programmer. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the value for the ITEM_NAME parameter is assigned in the body of the program.

Setting Variable to Initial Values:

```
DCL
      DATA_SET_NAME              CHAR(17) INIT('EMP'),
      STRING_EXPR                  CHAR(10) VAR
```

Library Declaration of Entry Points:

```
DECLARE DBGETKANJI ENTRY (CHARACTER(*),
                          CHARACTER(*),
                          CHARACTER(*) VARYING)
   RETURNS (BIT 48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLGETKANJI');
```

Invocation of Entry Points:

```
ITEM_NAME = "EMP_KNAME";
RSLT:= DBGETKANJI(DATA_SET_NAME,ITEM_NAME,STRING_EXPR) ;
IF RSLT^= 0
   THEN CALL EXCEPTIONHANDLER() ;
```

ALGOLGETREAL

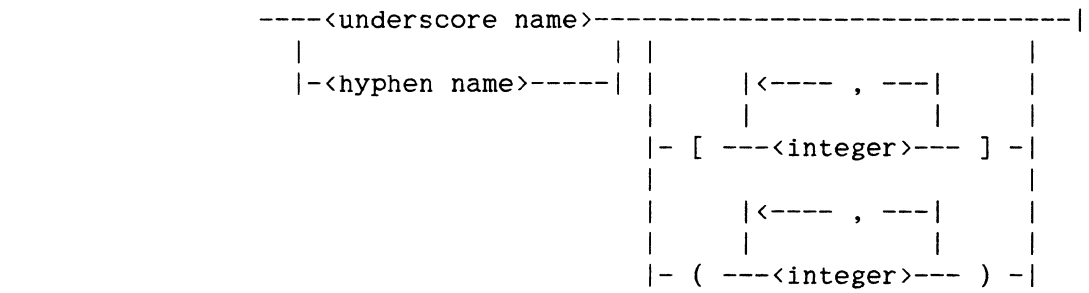
ALGOLGETREAL sets a variable declared in the program to the numeric value of a NUMBER, REAL, FIELD, COUNT, RECORDTYPE, or POPULATION data item in the current record of the specified data base structure. If an exception occurs, the variable is set to HIGH-VALUE (all bits ON), and an appropriate exception is returned. Numeric items that are too large to be retrieved by ALGOLGETREAL can be retrieved using the ALGOLGETDOUBLE entry point.

Required Arguments:	Data Types of Arguments:
<structure>	CHARACTER
<item name>	CHARACTER
<variable>	BINARY FLOAT(38)

Syntax

<structure>

<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

A PL/I variable declared as BINARY FLOAT(38).

Example

The following program fragments show how ALGOLGETREAL is used to move a data item value from a DMSII user work area to a variable declared by the programmer. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the value for the ITEM_NAME argument is assigned in the body of the program.

Setting Variables to Initial Values:

```
DCL
      DATA_SET_NAME           CHAR(17) INIT('EMP'),
      FLOAT_EXPR                FLOAT BINARY(38)
      ;
```

Library Declaration of Entry Point:

```
DECLARE DBGETFLOAT ENTRY (CHARACTER(*),
                          CHARACTER(*),BINARY FLOAT(38)
RETURNS (BIT(48))
OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLGETREAL');
```

Invocation of Entry Point:

```
ITEM_NAME = 'EMP-NO' ;
RSLT = DBGETFLOAT(DATA_SET_NAME,ITEM_NAME,FLOAT_EXPR) ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER() ;
```

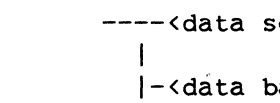
ALGOLGETBOOLEAN

ALGOLGETBOOLEAN retrieves a BOOLEAN value from a BOOLEAN item or a bit in a field item, and sets a PL/I variable to either '1'B for "TRUE" or '0'B for "FALSE" depending on the value. (PL/I's SUBSTR function may be used to isolate the right-most bit in the string.) The result descriptor will be set to a false indication if an exception occurs. The bit string that is returned will be truncated if the declared variable is too short.

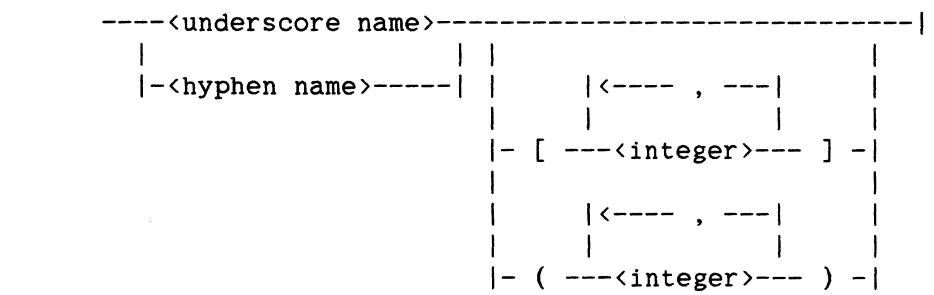
Required Arguments:	Data Types of Arguments:
<structure>	CHARACTER
<item name>	CHARACTER
<variable>	BIT(48)

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

A PL/I bit string containing 48 bits.

Example

The following program fragments show how ALGOLGETBOOLEAN is used to move a data item value from the user work area to a variable declared by the programmer. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the value for the ITEM_NAME argument is assigned in the body of the program.

Setting Variables to Initial Values:

```
DCL
      DATA_SET_NAME              CHAR(17) INIT('EMP'),
      BOOLEAN_EXPR               BIT(48)
      ;
```

Library Declaration of Entry Point:

```
DECLARE DBGETBIT ENTRY (CHARACTER(*),
                        CHARACTER(*), BIT(48))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI,ACTUALNAME = 'ALGOLGETBOOLEAN')
```

Invocation of Entry Point:

```
ITEM_NAME = 'EMP-SEC-CLEAR' ;
RSLT = DBGETBIT(DATA_SET_NAME,ITEM_NAME,BOOLEAN_EXPR) ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER() ;
```

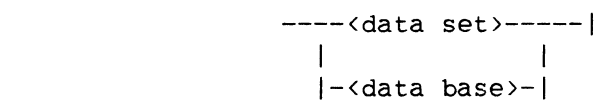
ALGOLGETDOUBLE

ALGOLGETDOUBLE retrieves numeric data items whose number of significant digits exceeds the capacity of a real variable. ALGOLGETDOUBLE sets a variable declared in the program to the numeric value of a NUMBER, REAL, FIELD, COUNT, RECORDTYPE, or POPULATION data item in the current record of the specified structure. If an exception occurs, the declared variable is set to HIGH-VALUE (all BITS on), and an appropriate exception is returned.

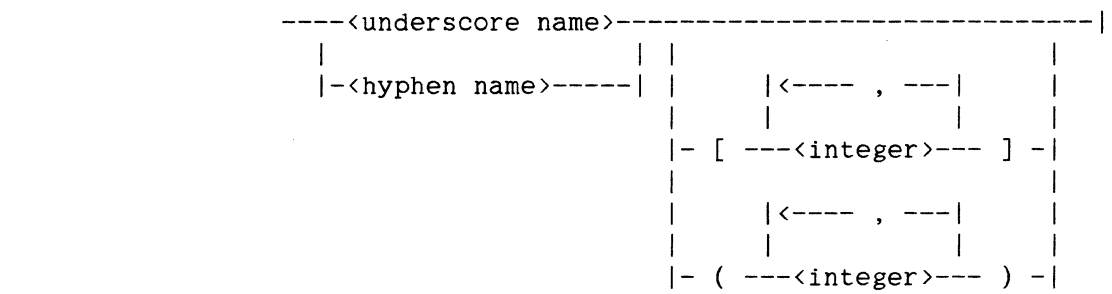
Required Arguments:	Data Types of Arguments:
<structure>	CHARACTER
<item name>	CHARACTER
<variable>	BINARY FLOAT(78)

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

A PL/I variable declared as BINARY FLOAT(78).

Example

The following program fragments show how ALGOLGETDOUBLE is used to move a data item value from a DMSII user work area to a variable declared by the programmer. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the value for the ITEM_NAME argument is assigned in the body of the program.

Setting Variables to Initial Values:

```
DCL
      DATA_SET_NAME              CHAR(17) INIT('EMP'),
      DFLOAT_EXPR                FLOAT BINARY(78)
      ;
```

Library Declaration of Entry Point:

```
DECLARE DBGETDOUBLE ENTRY (CHARACTER(*), CHARACTER(*),
                           BINARY FLOAT(78))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLGETDOUBLE')
```

Invocation of Entry Point:

```
ITEM_NAME = 'EMP-SSN' ;
RSLT = DBGETDOUBLE(DATA_SET_NAME,ITEM_NAME,DFLOAT_EXPR) ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER() ;
```

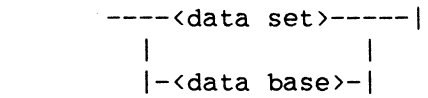
ALGOLPUTSTRING

ALGOLPUTSTRING puts a string into a particular data item in the current record of the specified structure. The data item must be declared in DASDL as an ALPHA or GROUP item. The string must not exceed the length of the data item where it is stored; otherwise, an exception is returned.

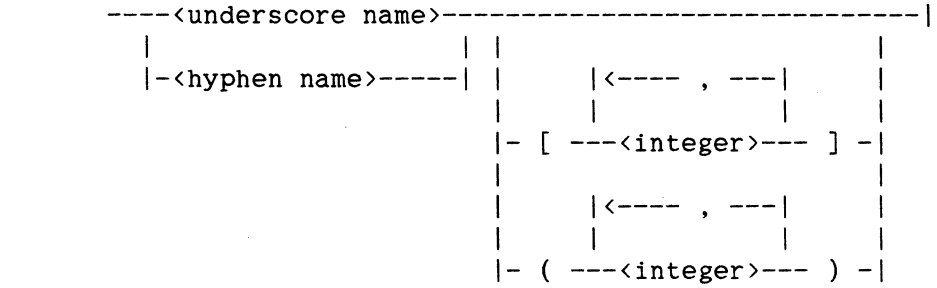
Required Arguments:	Data Types of Arguments:
<structure>	CHARACTER
<item name>	CHARACTER

Syntax

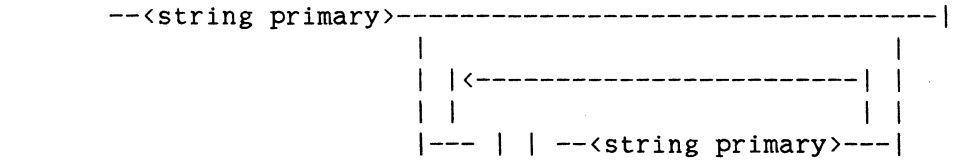
<structure>



<item name>



<string expression>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

<string expression>

A PL/I character string expression that evaluates to a string which will be put into the data base. For more information, refer to the PL/I Reference Manual.

Example

The following program fragments show how ALGOLPUTSTRING is used to move a data item value from a character string expression declared by the programmer to a DMSII user work area. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the values for the ITEM_NAME and STRING_EXPR arguments are assigned in the body of the program. Concatenation has also been used to create a string for the employee name being put into the data base.

Setting Variables to Initial Values:

```
DCL
      DATA_SET_NAME          CHAR(17) INIT('EMP')
      ;
```

Library Declaration of Entry Point:

```
DECLARE DBPUTCHARACTER ENTRY (CHARACTER(*),
                              CHARACTER(*), CHARACTER(*))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLPUTSTRING')
```

Invocation of Entry Point:

```
ITEM_NAME = 'EMP-FNAME' ;
STRING_EXPR = 'PEG' ;
RSLT = DBPUTCHARACTER(DATA_SET_NAME,ITEM_NAME,
                      STRING_EXPR ||  'GY') ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER() ;
```

ALGOLPUTKANJI

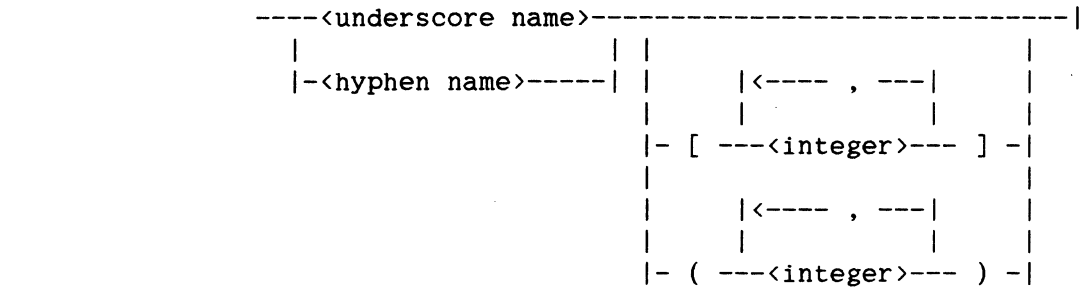
ALGOLPUTKANJI puts a string into a particular data item in the current record of the specified structure. The data item must be declared in DASDL as an ALPHA item of which USAGE IS KANJI. The string must not exceed the length of the data item where it will be stored; otherwise, an exception is returned.

Required Parameters:	Data Type of Parameters:
<structure>	CHARACTER
<item name>	CHARACTER
<variable>	CHARACTER

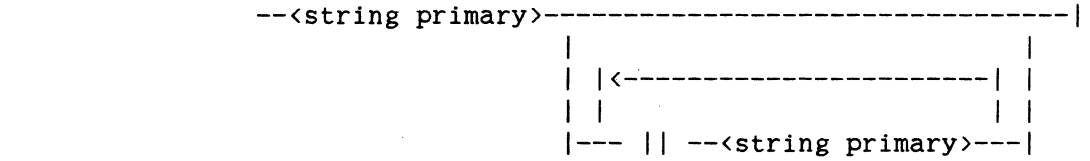
Syntax

<structure>

<item name>



<string expression>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<string expression>

A PL/I character string expression that evaluates to a string which will be put into the data base. For more information, refer to the PL/I Reference Manual.

<variable>

A PL/I character string variable.

Example

The following program fragments show how ALGOLGETKANJI is used to move a data item value from the DMSII user work area to a variable declared by the programmer. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the value for the ITEM_NAME parameter is assigned in the body of the program.

Setting Variable to Initial Values:

```
DCL
      DATA_SET_NAME              CHAR(17) INIT('EMP'),
      STRING_EXPR                  CHAR(10) VAR
```

Library Declaration of Entry Points:

```
DECLARE DBGETKANJI ENTRY (CHARACTER(*),
                          CHARACTER(*),
                          CHARACTER(*) VARYING)
   RETURNS (BIT 48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLGETKANJI');
```

Invocation of Entry Points:

```
ITEM_NAME = "EMP_KNAME";
RSLT:= DBGETKANJI(DATA_SET_NAME,ITEM_NAME,STRING_EXPR) ;
IF RSLT^= 0
   THEN CALL EXCEPTIONHANDLER() ;
```

ALGOLPUTREAL

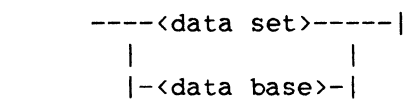
ALGOLPUTREAL puts a real value into a particular data item in the current record of the specified structure. The item must have been declared in the DASDL as a NUMBER, REAL, or FIELD item. If the argument for the real value produces an incorrect new value, an exception is returned.

If the number of significant digits in an item exceeds the capacity of a real expression, an ALGOL program should call the ALGOLPUTDOUBLE entry point to put real values into items of this type.

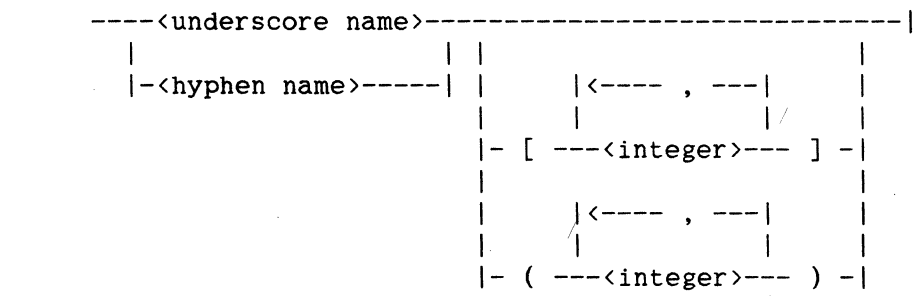
Required Arguments:	Data Types of Arguments:
<structure>	CHARACTER
<item name>	CHARACTER
<arithmetic expression>	BINARY FLOAT(38)

Syntax

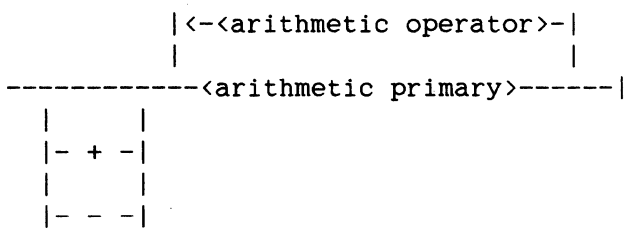
<structure>



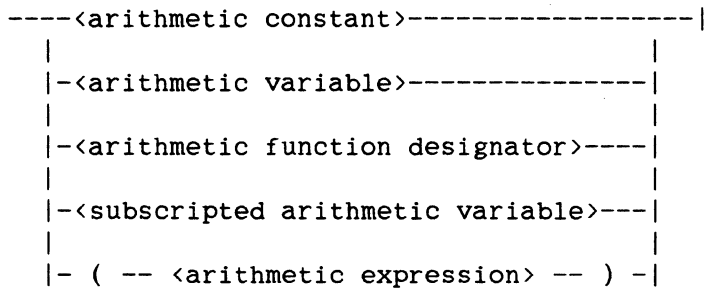
<item name>



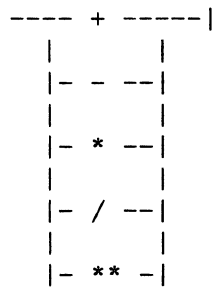
<arithmetic expression>



<arithmetic primary>



<arithmetic operator>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

<arithmetic expression>

A PL/I arithmetic expression that evaluates to a real value which will be put into the data base. For more information, refer to the PL/I Reference Manual.

Example

The following program fragments show how DBPUTREAL is used to move a data item value from a real expression declared by the programmer to a DMSII user work area. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the values for the ITEM_NAME and FLOAT_EXPR arguments are assigned in the body of the program.

Setting Variables to Initial Values:

```
DCL
      DATA_SET_NAME            CHAR(17) INIT('EMP')
      ;
```

Library Declaration of Entry Point:

```
DECLARE DBPUTFLOAT ENTRY (CHARACTER(*),
                          CHARACTER(*), BINARY FLOAT(38)
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLPUTREAL');
```

Invocation of Entry Point:

```
ITEM_NAME = 'EMP-NO' ;
FLOAT_EXPR = 11 ;
RSLT = DBPUTFLOAT(DATA_SET_NAME,ITEM_NAME,FLOAT_EXPR) ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER() ;
```

ALGOLPUTDOUBLE

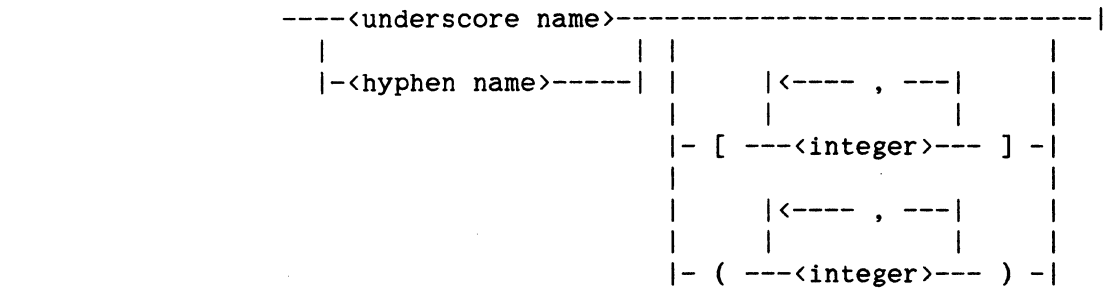
ALGOLPUTDOUBLE puts a double-precision real value into a data item in the current record of the specified structure. The data item must be a NUMBER, REAL, or FIELD item. If the arithmetic expression produces an incorrect new value, an exception is returned.

Required Arguments:	Data Types of Arguments:
<structure>	CHARACTER
<item name>	CHARACTER
<arithmetic expression>	BINARY FLOAT(38)

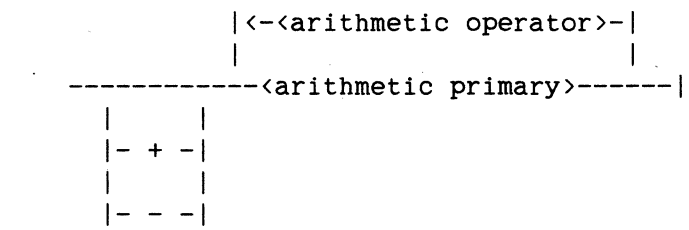
Syntax

<structure>

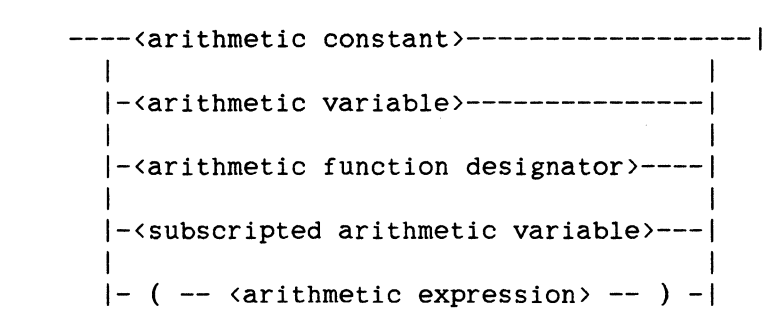
<item name>



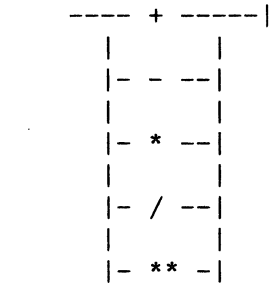
<arithmetic expression>



<arithmetic primary>



<arithmetic operator>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

<arithmetic expression>

A PL/I expression that evaluates to a double-precision real value which will be put into the data base. For more information, refer to the PL/I Reference Manual.

Example

The following program fragments show how ALGOLPUTDOUBLE is used to move a data item value from a numeric variable declared by the programmer to a DMSII user work area. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the values for the ITEM_NAME and DFLOAT_EXPR arguments are assigned in the body of the program. The arithmetic expression used is (DFLOAT_EXPR - 11).

Setting Variables to Initial Values:

```
DCL
      DATA_SET_NAME            CHAR(17) INIT('EMP')
      ;
```

Library Declaration of Entry Point:

```
DECLARE DBPUTDOUBLE ENTRY (CHARACTER(*), CHARACTER(*),
                           BINARY FLOAT(78))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLPUTDOUBLE')
```

Invocation of Entry Point:

```
ITEM_NAME = 'EMP-SSN' ;
DFLOAT_EXPR = 573230911 ;
RSLT = DBPUTDOUBLE(DATA_SET_NAME,
                   ITEM_NAME,DFLOAT_EXPR - 11) ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER() ;
```

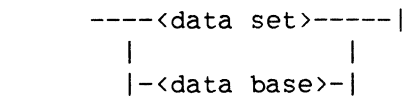
ALGOLPUTBOOLEAN

ALGOLPUTBOOLEAN puts a Boolean value into a particular BOOLEAN data item. The argument used to carry the Boolean value must be a bit string expression. For PL/I, '1'B or '0'B (representing TRUE or FALSE, respectively) should be placed in the right-most position of the bit string expression argument.

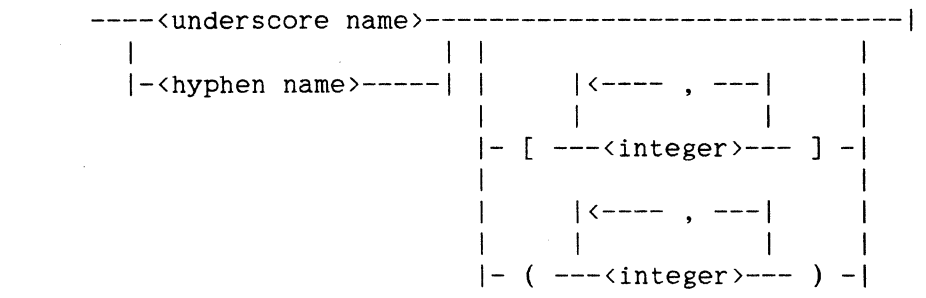
Required Arguments:	Data Types of Arguments:
<structure>	CHARACTER
<item name>	CHARACTER
<bit string expression>	BIT(48)

Syntax

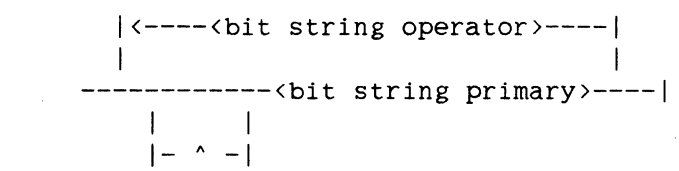
<structure>



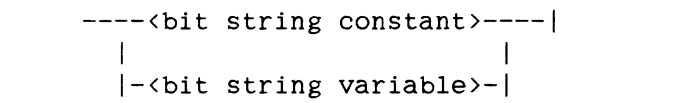
<item name>



<bit string expression>



<bit string primary>



<bit string operator>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

<bit string expression>

A PL/I expression that evaluates to '1'B for TRUE or '0'B for FALSE. For more information, refer to PL/I Reference Manual.

Example

The following program fragments show how ALGOLPUTBOOLEAN is used to move a bit string expression evaluating to FALSE from a PL/I program to a DMSII user work area. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the values for the ITEM_NAME and BOOLEAN_EXPR arguments are assigned in the body of the program.

Setting Variables to Initial Values:

```
DCL
      DATA_SET_NAME           CHAR(17) INIT('EMP')
      ;
```

Library Declaration of Entry Point:

```
DECLARE DBPUTBIT ENTRY (CHARACTER(*),
                        CHARACTER(*), BIT(48))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLPUTBOOLEAN')
```

Invocation of Entry Point:

```
ITEM_NAME = 'EMP-SEC-CLEAR' ;
BOOLEAN_EXPR = '1'B ;
RSLT = DBPUTBIT(DATA_SET_NAME,ITEM_NAME,BOOLEAN_EXPR) ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER() ;
```

ALGOLPUTNULL

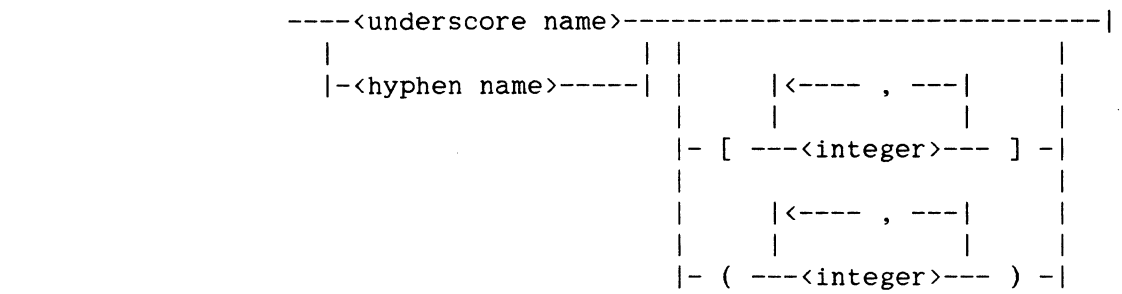
ALGOLPUTNULL sets a particular data item in the current record of the specified structure to its NULL value. The data item chosen should not be of type BOOLEAN, COUNT, RECORDTYPE, or POPULATION.

Required Arguments:	Data Types of Arguments:
<structure>	CHARACTER
<item name>	CHARACTER

Syntax

<structure>

<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

Example

The following program fragments show how ALGOLPUTNULL is used to set a data base item to its NULL value. In this example, a value for DATA_SET_NAME has been initialized at the beginning of the program, whereas the value for the ITEM_NAME argument is assigned in the body of the program.

Setting Variables to Initial Values:

```
DCL
      DATA_SET_NAME            CHAR(17) INIT('EMP')
      ;
```

Library Declaration of Entry Point:

```
DECLARE DBPUTNULL ENTRY (CHARACTER(*), CHARACTER(*))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLPUTNULL')
```

Invocation of Entry Point:

```
ITEM_NAME = 'EMP-SALARY' ;
RSLT = DBPUTNULL(DATA_SET_NAME,ITEM_NAME) ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER() ;
```

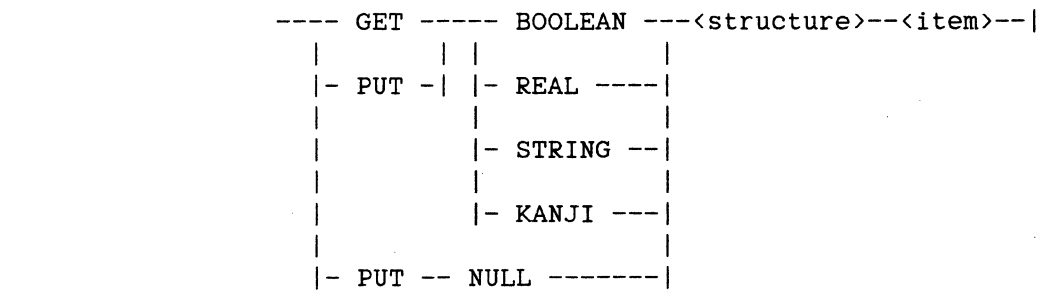
ALGOLDATA

ALGOLDATA allows "GET" and "PUT" commands to be dynamically constructed during program execution. The data request argument contains a "GET" or "PUT" operation. The data argument receives the value for "GET" operations and is used to hold the new value for "PUT" operations.

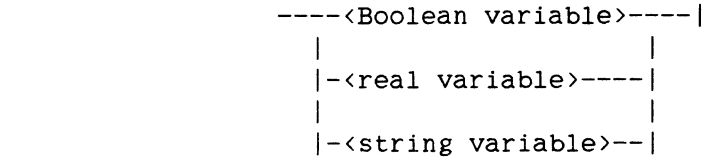
Required Arguments:	Data Types of Arguments:
<data request>	CHARACTER
<data>	CHARACTER VARYING

Syntax

<data request>



<data>



Explanation

GET <data type> <structure> <item>

Specifies that a data value of type BOOLEAN, REAL, STRING, or KANJI will be moved from an item in a specified data base structure to a variable declared by the programmer.

PUT <data type> <structure> <item>

Specifies that a data value of type BOOLEAN, REAL, STRING, KANJI, or NULL will be moved from a variable declared by the programmer to an item in a specified data base structure.

<data>

For PUT operations, this argument holds data that will be moved to the data base. For GET operations, this argument receives data from the data base.

Example

The following program fragments show how ALGOLDATA is used to move data from an ALGOL program to the data base. The type of operation being used, the type of item being moved, the data set name, the item name, and the data itself are all specified within the body of the program.

Setting Variables to Initial Values:

Not applicable when using DBDATA.

Library Declaration of Entry Point:

```
DECLARE DBDATA ENTRY (CHARACTER(*), CHARACTER(*) VARYING
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLDATA');
```

Invocation of Entry Point:

```
DATA_REQUEST = 'PUT REAL EMP EMP-DATE-HIRED' ;
DATA_BUFFER = '811223' ;
RSLT = DBDATA(DATA_REQUEST,DATA_BUFFER) ;
IF RSLT ^= 0
   THEN CALL EXCEPTIONHANDLER() ;
```

5.3 EXCEPTION-HANDLING ENTRY POINTS

The exception-handling entry points are among the functions of the interpretive interface that are allowed even when the data base is closed.

All PL/I entry points of the DMINTERPRETER Library except ALGOLEXCEPTIONTEXT and ALGOLEXCEPTIONNAME return a particular value to indicate that an exception has occurred. After a PL/I entry point call to the DMINTERPRETER Library has been completed, the entry point returns an integer to the result descriptor in the PL/I invocation procedure. The returned value is 1 if an exception has occurred and 0 if no exception has occurred. A PL/I exception is the standard DMSII exception word which includes the exception category and subcategory numbers and the number of the structure involved.

The DBSTATUS entry point can be subsequently called to retrieve the actual exception, which is the same exception as that returned by the last entry point called. The exception result returned by ALGOLSTATUS can be passed to ALGOLEXCEPTIONNAME and/or ALGOLEXCEPTIONTEXT to find out the name of the exception or the full exception message, respectively.

When the ALGOLEXCEPTIONTEXT or ALGOLEXCEPTIONNAME entry point is called, the DMINTERPRETER Library calls the DMSUPPORT Library to retrieve the text or the name of the exception message. The ALGOLEXCEPTIONTEXT or ALGOLEXCEPTIONNAME entry points can return any exception that can be returned by the DMSII user language interface. In addition, the DMSUPPORT Library contains a special category of exceptions called INTLIBERRORS, for errors peculiar to the interpretive interface. (Refer to the appendix "DMSII Exceptions and Errors" for a list of exception categories and subcategories and the corresponding exception messages.)

ALGOLSTATUS

ALGOLSTATUS returns the exception result of the last entry point call that was made. The program does not need to directly examine the contents of of this exception result, but should pass the result on to ALGOLEXCEPTIONNAME or ALGOLEXCEPTIONTEXT, to find out the exception message or exception name from the DMSUPPORT Library.

Required Arguments:	Data Types of Arguments:
None for PL/I.	Not applicable.

Example

The following program fragments show how ALGOLSTATUS is used to obtain an exception result from the last entry point that was called.

Setting Variables to Initial Values:

```
DCL
      STATUSRSLT                BIT(48)
      ;
```

Library Declaration of Entry Point:

```
DECLARE DBSTATUS ENTRY
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLSTATUS');
```

Invocation of Entry Point:

```
STATUSRSLT = DBSTATUS ;
```

ALGOLEXCEPTIONTEXT

A call on ALGOLEXCEPTIONTEXT should be preceded by a call on ALGOLSTATUS. After ALGOLSTATUS returns the exception-word for the latest call on the DMINTERPRETER Library, ALGOLEXCEPTIONTEXT can return the text of the exception message in its text argument.

The exception message can later be displayed by the program. The text argument contains "NO EXCEPTION" if no exception has occurred. If text is not long enough to hold the entire message, the message is truncated to fit in the available space. If the space is longer than the message text, then it is padded with trailing blanks.

If an error message is encountered when ALGOLEXCEPTIONTEXT is called, the user should investigate whether the DMSUPPORT Library is currently unavailable, possibly for security reasons.

Required Arguments:	Data Types of Arguments:
<exception>	BIT(48)
<text>	CHARACTER VARYING

Explanation

<exception>

A bit string variable containing the exception information from a DBSTATUS entry point call.

<text>

A character string variable with enough room to hold error messages without truncation if possible.

Example

The following procedure shows how ALGOLEXCEPTIONTEXT is used to obtain the exception message that corresponds to the exception word returned by the ALGOLSTATUS entry point.

Library Declaration of Entry Point:

```
DECLARE DBEXCEPTIONTEXT ENTRY (BIT(48),
                               CHARACTER(*) VARYING)
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI,
            ACTUALNAME = 'ALGOLEXCEPTIONTEXT');
```

Exception-Handling Procedure:

```
EXCEPTIONHANDLER:PROCEDURE ;

DCL
    STATUSRSLT          BIT(48)
    ;

    STATUSRSLT = DBSTATUS ;
    CALL DBEXCEPTIONNAME(STATUSRSLT,EXCEPTIONMSG);
    PUT SKIP LIST('EXCEPTION-NAME=',EXCEPTIONMSG) ;

    CALL DBEXCEPTIONTEXT(STATUSRSLT,EXCEPTIONMSG);
    PUT SKIP LIST('EXCEPTION-TEXT=',EXCEPTIONMSG) ;

END EXCEPTIONHANDLER ;
```

ALGOLEXCEPTIONNAME

A call on ALGOLEXCEPTIONNAME should be preceded by a call on ALGOLSTATUS. For an exception returned by ALGOLSTATUS, ALGOLEXCEPTIONNAME returns in the text argument the name of the type of exception that occurred. The text contains either the name of a DMSII exception category, such as NOTFOUND or OPENERROR, or "NO EXCEPTION" if no exception has occurred. If text is not long enough to hold the entire message, the message is truncated to fit in the available space. If text is longer than the message, the extra space is padded with trailing blanks.

If an error message is encountered when ALGOLEXCEPTIONTEXT is called, the user should investigate whether the DMSUPPORT Library is currently unavailable, possibly for security reasons.

Required Arguments:	Data Types of Arguments:
<exception>	BIT(48)
<text>	CHARACTER VARYING

Explanation

<exception>

A bit string variable containing the exception information from a DBSTATUS entry point call.

<text>

A character string variable with enough room to hold error messages without truncation if possible.

Example

The following procedure shows how ALGOLEXCEPTIONNAME is used to obtain the name of the exception that corresponds to the exception result returned by the ALGOLSTATUS entry point.

Library Declaration of Entry Point:

```
DECLARE DBEXCEPTIONNAME ENTRY (BIT(48),
                               CHARACTER(*) VARYING)
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI,
            ACTUALNAME = 'ALGOLEXCEPTIONNAME');
```

Exception-Handling Procedure:

```
EXCEPTIONHANDLER:PROCEDURE ;

DCL
    STATUSRSLT          BIT(48)
    ;

    STATUSRSLT = DBSTATUS ;
    CALL DBEXCEPTIONNAME(STATUSRSLT,EXCEPTIONMSG);
    PUT SKIP LIST('EXCEPTION-NAME=',EXCEPTIONMSG) ;

    CALL DBEXCEPTIONTEXT(STATUSRSLT,EXCEPTIONMSG);
    PUT SKIP LIST('EXCEPTION-TEXT=',EXCEPTIONMSG) ;

END EXCEPTIONHANDLER ;
```

5.4 ATTRIBUTE-SETTING ENTRY POINTS

ALGOLSETLIMIT, the only PL/I attribute-setting entry point, is used to restrict the number of times the DMINTERPRETER can call the Accessroutines to complete a find. FINDLIMIT is the only valid limit type allowed for the ALGOLSETLIMIT entry point.

ALGOLSETLIMIT can be used to declare DMINTERPRETER entry points in application programs, unless renaming is desired or necessary due to limitations of a particular language.

ALGOLSETLIMIT

ALGOLSETLIMIT sets the various limit values for the DMINTERPRETER.

Required Parameters:	Data Type of Parameters:
<limittype>	CHARACTER
<limitvalue>	CHARACTER

Explanation

<limittype>=FINDLIMIT

A string parameter which specifies that type of limit to be set, and specifies the maximum number of times the DMINTERPRETER can call the Accessroutines to perform a FIND or LOCK.

FINDLIMIT is the only <limittype> currently defined. By setting FINDLIMIT the user can prevent long linear searches.

If an invalid <limittype> is specified, INTLIBERROR 33 is returned. If the DMINTERPRETER exceeds the FINDLIMIT while performing a FIND or LOCK, INTLIBERROR 35 is returned.

<limitvalue>

An integer string parameter which specifies the value to be assigned to <limittype>. Negative values are not allowed for <limittype>. Specifying a <limitvalue> of 0 (zero) disables the FINDLIMIT mechanism, allowing infinitely long linear searches.

When the DMINTERPRETER is first initialized, FINDLIMIT is set to the default value of 0 (zero).

6 ENTRY POINTS IN FORTRAN77

This section explains the FORTRAN77 interface to the DMINTERPRETER Library, and provides a sample application program written in FORTRAN77 to illustrate the following concepts:

1. How entry points of the DMINTERPRETER Library are declared in a FORTRAN77 application program.

2. How a FORTRAN77 application program "calls" or "invokes" entry points of the DMINTERPRETER Library.

3. How the DMINTERPRETER Library returns results to FORTRAN77 programs.

4. How exceptions returned by entry point calls are handled.

This section also lists all entry points of the DMINTERPRETER Library for FORTRAN77 and describes their functions and arguments. The following information is included under each description of an entry point:

1. A brief explanation of the function of the entry point.

2. Syntax diagrams and semantics that describe required arguments. (More detailed syntax diagrams for data base constructs used with the interpretive interface are given in the appendix "Syntax Diagrams for Entry Point Parameters.")

3. A program fragment that illustrates the declaration of the entry point within an application program.

4. A program fragment that shows how variables for entry point parameters are set to initial values within an application program.

5. A program fragment that shows how the entry point is invoked.

Table 6-1 summarizes the FORTRAN77 entry points, and lists them in the order in which they are described later in this section.

Table 6-1. Summary
of FORTRAN77 Entry Points

Standard Entry Points:

FORTRAN77OPEN
FORTRAN77CLOSE
FORTRAN77BEGINTRANSACTION
FORTRAN77ENDTRANSACTION
FORTRAN77CREATE
FORTRAN77RECREATE
FORTRAN77STORE
FORTRAN77FREE
FORTRAN77SET
FORTRAN77FIND
FORTRAN77LOCK
FORTRAN77DELETE
FORTRAN77VERB

Data Transfer Entry Points:

FORTRAN77GETCHARACTER
FORTRAN77GETKANJI
FORTRAN77GETREAL
FORTRAN77GETDOUBLE
FORTRAN77GETLOGICAL
FORTRAN77PUTCHARACTER
FORTRAN77PUTKANJI
FORTRAN77PUTREAL
FORTRAN77PUTDOUBLE
FORTRAN77PUTLOGICAL
FORTRAN77PUTNULL
FORTRAN77DATA

Exception-Handling Entry Points:

FORTRAN77STATUS
FORTRAN77EXCEPTIONTEXT
FORTRAN77EXCEPTIONNAME

Attribute-Setting Entry Point:

FORTRAN77SETLIMIT

LIBRARY DECLARATION

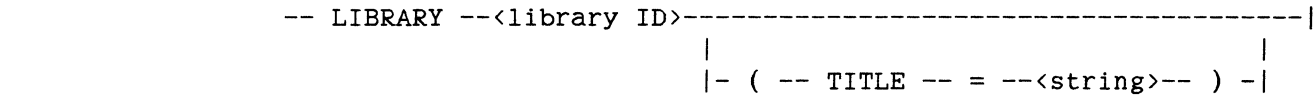
The first step in writing an application program that accesses the interpretive interface is to declare the DMINTERPRETER Library and its entry points within the application program. Linkage to the DMINTERPRETER Library must be direct, because the library actually contains the procedures that are named in its EXPORT declarations.

In FORTRAN77, a BLOCK GLOBALS subprogram is used for declaring libraries. Only one BLOCK GLOBALS subprogram is allowed in a FORTRAN77 program, and it must precede all other FORTRAN77 source statements. The BLOCK GLOBALS subprogram is initiated by a BLOCK GLOBALS statement and terminated by an END statement.

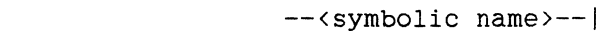
Syntax for the FORTRAN77 library statement, which must appear in the BLOCK GLOBALS Subprogram, is as follows:

Syntax

<library statement>



<library ID>



Explanation

In the library statement, the external name of the library code file (which has been compiled for a particular data base) is assigned to the library identifier by using the TITLE attribute. The library identifier is used as an internal name within the application program to refer to the DMINTERPRETER Library.

The library statement declares a symbolic name to be a library identifier. A FORTRAN77 symbolic name is a sequence of one to six letters or digits, the first of which must be a letter. In the TITLE clause, the string that names the library code file is a nonempty string of no more than 255 characters which may be delimited with an apostrophe

or quote. (For more information on FORTRAN77 basic constructs, refer to the FORTRAN77 Programming Reference Manual.)

In the sample FORTRAN77 program provided in this section of the manual, the library statement is used in a BLOCK GLOBALS subprogram as follows:

```
BLOCK GLOBALS
   LIBRARY DMI(TITLE="DMINTERPRETER/EMPJOB.")
END
```

ENTRY POINT DECLARATION

To ease the process of writing programs that use the DMINTERPRETER Library, the DATABASE/DMINTERPRETER symbolic file provided on the release tapes contains a declaration of the library and its entry points, as well as type declarations for the entry points, for the FORTRAN77 language. A FORTRAN77 program must explicitly declare a library called DMI in its library statement, if these entry point declarations are used.

FORTRAN77 requires the library entry point declarations to be included once at the beginning of every program, and the type declarations to be included in every subprogram within the program. These declarations can be included in application programs by using the $ INCLUDE compiler control option for the following sequence ranges within DATABASE/DMINTERPRETER:

Library Entry Point Declarations:	20200000 - 20249999
Type Declarations:	20290000 - 20299999

The includable entry points rename exported entry points as follows:

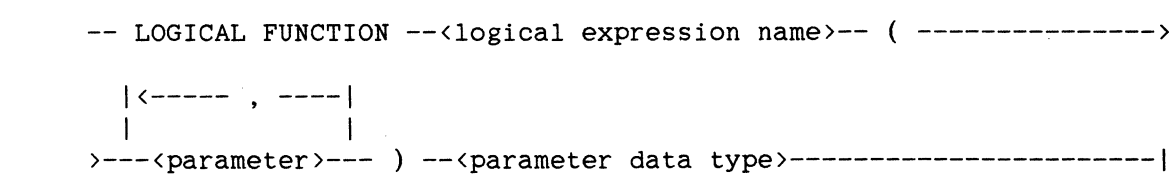
Exported Name of Entry Point:	Renamed As:
FORTRAN77OPEN	DBOPEN
FORTRAN77CLOSE	DBCLOSE
FORTRAN77BEGINTRANSACTION	DBBTR
FORTRAN77ENDTRANSACTION	DBETR
FORTRAN77CREATE	DBCR
FORTRAN77RECREATE	DBRCR
FORTRAN77STORE	DBSTORE
FORTRAN77FREE	DBFREE
FORTRAN77SET	DBSET
FORTRAN77FIND	DBFIND
FORTRAN77LOCK	DBLOCK
FORTRAN77DELETE	DBDEL
FORTRAN77VERB	DBVERB
FORTRAN77GETCHARACTER	DBGETC
FORTRAN77GETKANJI	DBGETK
FORTRAN77GETREAL	DBGETR
FORTRAN77GETDOUBLE	DBGETD
FORTRAN77GETLOGICAL	DBGETL
FORTRAN77PUTCHARACTER	DBPUTC
FORTRAN77PUTKANJI	DBPUTK
FORTRAN77PUTREAL	DBPUTR
FORTRAN77PUTDOUBLE	DBPUTD
FORTRAN77PUTLOGICAL	DBPUTL
FORTRAN77PUTNULL	DBPUTN
FORTRAN77DATA	DBDATA

Exported Name of Entry Point:	Renamed As:
FORTRAN77STATUS	DBSTAT
FORTRAN77EXCEPTIONTEXT	DBEXCT
FORTRAN77EXCEPTIONNAME	DBEXCN
FORTRAN77SETLIMIT	DBSETL

Data types for the parameters of each entry point are given under individual entry point descriptions later in this section. The includable sequence range declares all entry points, renaming them with the "DB" prefix. Users who want to rename the exported FORTRAN77 entry points in a different manner, or declare only a subset of the entry points, must individually declare the entry points within the application program.

The following diagram shows how a library entry point is declared in FORTRAN77:

Syntax

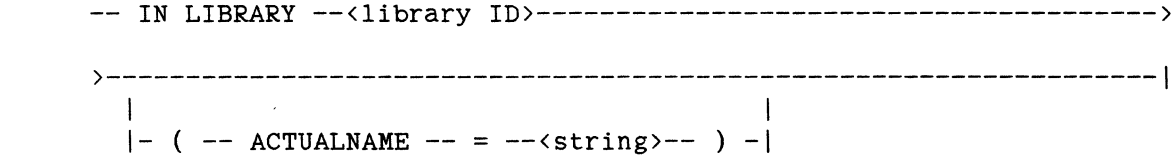


Explanation

Associating an entry point's logical expression name with the library that is known to the application program through the library ID is accomplished with the IN LIBRARY statement:

Syntax

<in library statement>



Explanation

The ACTUALNAME clause allows the renaming of a library entry point by using a different symbolic name in the library entry point declaration which can still link to the appropriate entry point through ACTUALNAME. The character string in the ACTUALNAME clause is the actual name that is given for this logical expression in the template of the library.

Example

The following example shows how a FORTRAN77 entry point has been declared in the includable sequence range of DATABASE/DMINTERPRETER:

Library Entry Point Declaration:

```
LOGICAL FUNCTION DBOPEN (OTYPE)
   CHARACTER OTYPE
   IN LIBRARY DMI (ACTUALNAME = "FORTRAN77OPEN")
END
```

INVOCATION OF ENTRY POINTS

In FORTRAN77, each entry point of the DMINTERPRETER Library is a logical expression. An entry point is invoked with a logical assignment statement, whereby the logical expression is evaluated and the resultant logical value is assigned to the logical variable. The following example shows a typical method of invoking a FORTRAN77 entry point of the DMINTERPRETER Library, where RSLT is a logical variable, and EXCEPT is a subroutine for handling exceptions that is referenced by a CALL statement:

How to Invoke an Entry Point:

```
RSLT = DBOPEN(UPDATE)
IF (RSLT) CALL EXCEPT
```

After an entry point call on the DMINTERPRETER Library, a result descriptor is returned as a logical value. This result descriptor is a 48-bit word which is the standard Data Management System II (DMSII) exception word; it includes the exception category and subcategory, and the structure number involved. When a call on the DMINTERPRETER Library causes an exception, the last bit in the 48-bit word is set to 1. (When a call on the library is successful, the 48-bit word contains all zeros.) For more information on exceptions, refer to "Exception-Handling Entry Points" later in this section.

6.1 STANDARD ENTRY POINTS

Standard entry points correspond to functions performed by DMSII user language interface statements. The results returned by standard entry points are used for exception handling.

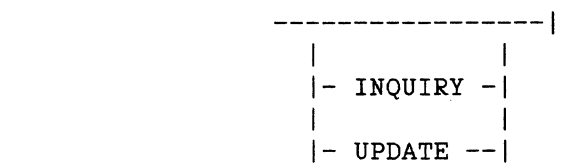
FORTRAN77OPEN

FORTRAN77OPEN opens the data base for inquiry or update, depending on which open disposition is used. FORTRAN77OPEN must be called before any data base operations are performed. FORTRAN77OPEN, the exception handling entry points (FORTRAN77STATUS, FORTRAN77EXCEPTIONTEXT, and FORTRAN77EXCEPTIONNAME), and operations on the intrinsic data set DBSTRUCTURE are the only functions allowed when the data base is closed.

Required Parameters:	Data Type of Parameters:
<open disposition>	CHARACTER

Syntax

<open disposition>



Explanation

INQUIRY

Causes the data base to be opened for INQUIRY. No updates are allowed.

UPDATE

Causes the data base to be opened for UPDATE. Both inquiry and update are allowed.

Example

The following program fragments show how FORTRAN77OPEN is used to open the data base for UPDATE. A variable must be set to an initial value of UPDATE or INQUIRY before invoking the entry point.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBOPEN (OTYPE)
   CHARACTER OTYPE
   IN LIBRARY DMI (ACTUALNAME = "FORTRAN77OPEN")
END
```

Setting Variables to Initial Values:

```
CHARACTER UPDATE      *6      /"UPDATE"/
```

Invoking the Entry Point:

```
RSLT = DBOPEN(UPDATE)
IF (RSLT) CALL EXCEPT
```

FORTRAN77CLOSE

FORTRAN77CLOSE explicitly closes the data base. (The data base is implicitly closed when the library execution completes.)

Required Parameters:	Data Type of Parameters:
None	Not applicable

Example

The following example shows how FORTRAN77CLOSE is used to close the data base. When invoking FORTRAN77CLOSE, no parameters need to be specified, because none are required for this entry point.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBCLOSE()
   IN LIBRARY DMI (ACTUALNAME = "FORTRAN77CLOSE")
END
```

Setting Variables to Initial Values:

Not applicable.

Invoking the Entry Point:

```
RSLT = DBCLOSE
IF (RSLT) CALL EXCEPT
```

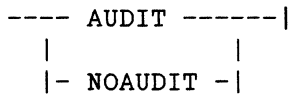
FORTRAN77BEGINTRANSACTION

FORTRAN77BEGINTRANSACTION causes the user program to enter transaction state. The audit parameter specifies whether an image of the current restart data set record is to be audited. Calls on FORTRAN77BEGINTRANSACTION are ignored for unaudited data bases.

Required Parameters:	Data Type of Parameters:
<audit>	CHARACTER

Syntax

<audit>



Explanation

AUDIT

Specifies that an image of the current restart data set will be audited.

NOAUDIT

Specifies that auditing an image of the current restart data set is not required. NOAUDIT is the default value of the audit parameter.

Example

The following example shows how a user program can enter transaction state by calling FORTRAN77BEGINTRANSACTION. A variable must be set to an initial value of AUDIT or NOAUDIT before invoking the entry point.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBBTR (AUDIT)
   CHARACTER AUDIT
IN LIBRARY DMI (ACTUALNAME =
  "FORTRAN77BEGINTRANSACTION")
END
```

Setting Variables to Initial Values:

```
CHARACTER AUDIT       *10      /"AUDIT"/
```

Invoking the Entry Point:

```
RSLT = DBBTR(AUDIT)
IF (RSLT) CALL EXCEPT
```

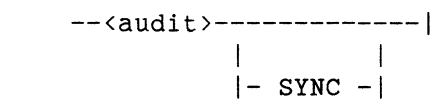
FORTRAN77ENDTRANSACTION

FORTRAN77ENDTRANSACTION causes the user program to leave transaction state. FORTRAN77ENDTRANSACTION should be used only with audited data bases. Calls on FORTRAN77ENDTRANSACTION are ignored for unaudited data bases.

Required Parameters:	Data Types of Parameters:
<auditsync>	CHARACTER

Syntax

<auditsync>



Explanation

<audit>

Specifies whether an image of the current restart data set will be audited (this is the audit parameter discussed under FORTRAN77BEGINTRANSACTION in this section).

<audit> SYNC

Forces a syncpoint, when an image of the current restart data set is audited.

Example

The following program fragments show how FORTRAN77ENDTRANSACTION is used to cause a program to exit transaction state, with the current restart data set record being audited. A variable must be set to an initial value of AUDIT or AUDIT SYNC before invoking the entry point.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBETR (AUDIT)
   CHARACTER AUDIT
   IN LIBRARY DMI (ACTUALNAME =
    "FORTRAN77ENDTRANSACTION")
END
```

Setting Variables to Initial Values:

```
AUDIT       *10      /"AUDIT"/
```

Invoking the Entry Point:

```
RSLT = DBETR(AUDIT)
IF (RSLT) CALL EXCEPT
```

FORTRAN77CREATE

FORTRAN77CREATE creates a new record for the specified data set. The record type parameter specifies whether the record will be fixed-length or variable-format.

Required Parameters:	Data Types of Parameters:
<data set>	CHARACTER
<record type>	CHARACTER

Syntax

<data set>

```
--<qualified name>--|
```

<record type>

```
-- " ---<integer>--- " --|
      |           |
      |-<blank>---|
```

Explanation

<data set>

Names a data set for which a new record will be created.

"<integer>"

A non-zero integer specifies that the record will be variable-length. If the specified integer is 0, the fixed part of a variable-length record is created.

"<blank>"

If an empty string is specified, a fixed-length record will be created.

Example

The following program fragments show how FORTRAN77CREATE is used to create a new fixed-length record belonging to a data set named EMP.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBCR (STR, RECTYP)
   CHARACTER STR, RECTYP
   IN LIBRARY DMI (ACTUALNAME = "FORTRAN77CREATE")
END
```

Setting Variables to Initial Values:

```
 CHARACTER DSNAME      *4      /"EMP"/,
*          SPACE      *1      /" "/
```

Invoking the Entry Point:

```
RSLT = DBCR(DSNAME,SPACE)
IF (RSLT) CALL EXCEPT
```

FORTRAN77RECREATE

FORTRAN77RECREATE re-creates, or copies, a new image of an existing record for the purpose of modification. The data set parameter names the existing data set where the record resides. The record type parameter specifies whether the re-created record will be fixed-length or variable-format.

Required Parameters:	Data Types of Parameters:
<data set>	CHARACTER
<record type>	CHARACTER

Syntax

<data set>

```
--<qualified name>--|
```

<record type>

```
-- " ---<integer>--- " --|
      |           |
      |-<blank>---|
```

Explanation

<data set>

Names a data set for which a record will be re-created.

"<integer>"

A non-zero integer specifies that the record will be variable-length. If the specified integer is 0 (zero), the fixed part of a variable-length record is created.

"<blank>"

If an empty string is specified, a fixed-length record will be created.

Example

The following program fragments show how FORTRAN77RECREATE is used to re-create an image of an existing fixed-length record belonging to a data set named EMP. To see all the steps that need to be taken in the re-creation of an existing record, refer to the "RE-CREATING A RECORD" fragment of the FORTRAN77 program in the appendix "Sample Programs."

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBRCR (STR, RECTYP)
   sho
   IN LIBRARY DMI (ACTUALNAME = "FORTRAN77RECREATE")
END
```

Setting Variables to Initial Values:

```
CHARACTER DSNAME      *4      /"EMP"/,
*         SPACE       *1      /" "/
```

Invoking the Entry Point:

```
RSLT = DBRECR(DSNAME,SPACE)
IF (RSLT) CALL EXCEPT
```

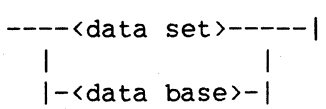
FORTRAN77STORE

FORTRAN77STORE stores a record in the specified structure. For audited data bases, FORTRAN77BEGINTRANSACTION must have been called before FORTRAN77STORE is called, because the program must be in transaction state in order for a record to be stored. If FORTRAN77STORE is called outside of transaction state for an audited data base, an exception is returned.

Required Parameters:	Data Types of Parameters:
<structure>	CHARACTER

Syntax

<structure>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

Example

The following program fragments show how FORTRAN77STORE is used to store a record in a data set named EMP.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBSTORE (STR)
   CHARACTER STR
   IN LIBRARY DMI (ACTUALNAME = "FORTRAN77STORE")
END
```

Setting Variables to Initial Values:

```
CHARACTER DSNAME      *4      /"EMP"/
```

Invoking the Entry Point:

```
RSLT = DBSTORE(DSNAME)
IF (RSLT) CALL EXCEPT
```

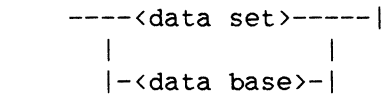
FORTRAN77FREE

FORTRAN77FREE unlocks a data set record or a global data record. FORTRAN77FREE may be used to unlock a record that you anticipate may not be implicitly freed for a long time. After FORTRAN77FREE is called for a record, other programs may lock that record.

Required Parameters:	Data Types of Parameters:
<structure>	CHARACTER

Syntax

<structure>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

Example

The following program fragments show how FORTRAN77FREE is used to unlock a record belonging to a specified structure. In this case, a record in the data set EMP is being unlocked. Variables should be initialized in advance to the appropriate values.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBFREE (STR)
   CHARACTER STR
   IN LIBRARY DMI (ACTUALNAME = "FORTRAN77FREE")
END
```

Setting Variables to Initial Values:

```
CHARACTER DSNAME      *4      /"EMP"/
```

Invoking the Entry Point:

```
RSLT = DBFREE(DSNAME)
IF (RSLT) CALL EXCEPT
```

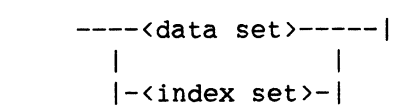
FORTRAN77SET

FORTRAN77SET sets particular data sets or index sets to their beginning or ending points, or sets the position pointer of an index set to the current record of a particular data set. The following syntax diagrams and semantics explain which structures can be set to which targets.

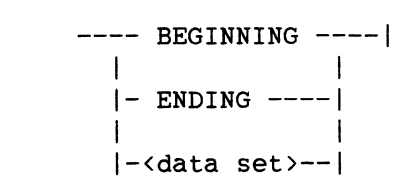
Required Parameters:	Data Type of Parameters:
<structure>	CHARACTER
<target>	CHARACTER

Syntax

<structure>



<target>



Explanation

<data set> BEGINNING

Names a particular data set that will be set to its beginning point at run time.

<data set> ENDING

Names a particular data set that will be set to its ending point at run time.

<index set> <data set>

Names the data set spanned by the index set, and sets the position pointer of the index set to the current record within the spanned data set.

<index set> BEGINNING

Names a particular index set that will be set to its beginning point at run time.

<index set> ENDING

Names a particular index set that will be set to its ending point at run time.

Example

The following program fragments show how FORTRAN77SET is used to set a particular data base structure to a desired target. In this case, the data set named EMP is set to its beginning point.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBSET (STR, TARGET)
   CHARACTER STR, TARGET
   IN LIBRARY DMI (ACTUALNAME = "FORTRAN77SET")
END
```

Setting Variables to Initial Values:

```
CHARACTER DSNAME      *4      /"EMP"/,
*         BEGIN       *9      /"BEGINNING"/
```

Invoking the Entry Point:

```
RSLT = DBSET(DSNAME,BEGIN)
IF (RSLT) CALL EXCEPT
```

FORTRAN77FIND

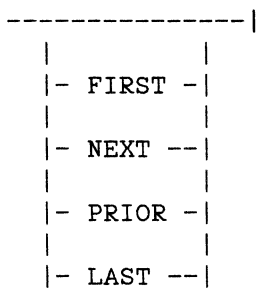
FORTRAN77FIND transfers a record to the user work area associated with the specified data base structure. The other parameters shown below indicate the direction in which the FIND operation proceeds, and the selection expression used during the search (provided that the condition parameter is non-empty).

An FORTRAN77FIND call is more sophisticated than a user language interface FIND statement. Several restrictions have been eased, and some algorithmic enhancements have been made. These improvements are discussed in detail in the appendix "DBFIND Enhancements."

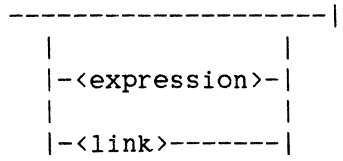
Required Parameters:	Data Type of Parameters:
<direction>	CHARACTER
<structure>	CHARACTER
<condition>	CHARACTER

Syntax

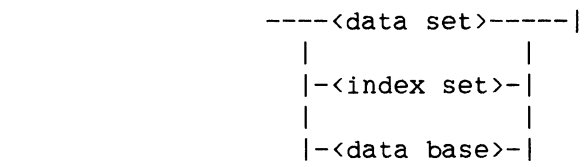
<direction>



<condition>



<structure>



Explanation

(empty)

If the direction parameter is empty, then the FIND operation will transfer the current record in the specified structure to the user work area.

FIRST

Indicates that the FIND operation will transfer the first record in the specified structure to the user work area.

NEXT

Indicates that the FIND operation will transfer the next record in the specified structure to the user work area.

PRIOR

Indicates that the FIND operation will transfer the prior record in the specified structure to the user work area.

LAST

Indicates that the FIND operation will transfer the last record in the specified structure to the user work area.

<data set>

Names a data set in the data base that will be accessed at run time.

<index set>

Names an index set in the data base that will be used for the search.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<expression>

A selection expression used during the DBFIND operation to find records that meet a specified condition.

<link>

Specifies the data base structure or data set item that links the current record to a record in another data set.

Example

The following program fragments show how FORTRAN77FIND is used to find the first record in a data set named EMP, which satisfies a specified condition.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBFIND (STR, ITEM, WHERE)
   CHARACTER STR, ITEM, WHERE
   IN LIBRARY DMI (ACTUALNAME = "FORTRAN77FIND")
END
```

Setting Variables to Initial Values:

```
 CHARACTER DSNAME      *4      /"EMP"/,
*          FIRST      *10     /"FIRST"/,
*          COND       *9      /"EMP-NO=11"/
```

Invoking the Entry Point:

```
RSLT = DBFIND(FIRST,DSNAME,COND)
IF (RSLT) CALL EXCEPT
```

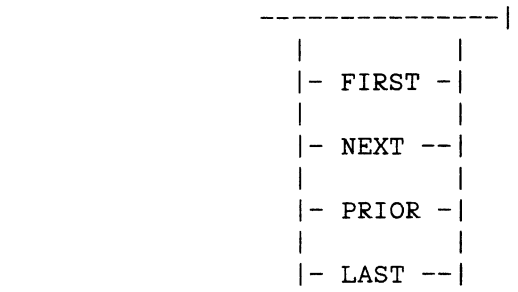
FORTRAN77LOCK

FORTRAN77LOCK functions in the same manner as FORTRAN77FIND, except that a record is locked after it is found. FORTRAN77LOCK transfers a record to the user work area associated with the specified data structure and locks it there for modification. The other parameters shown below indicate the direction in which the search for the record proceeds, and the selection expression used during the search.

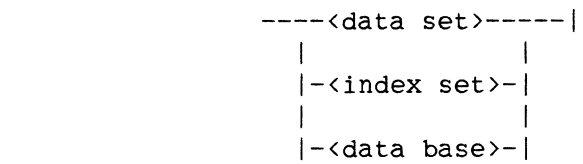
Required Parameters:	Data Type of Parameters:
<direction>	CHARACTER
<structure>	CHARACTER
<condition>	CHARACTER

Syntax

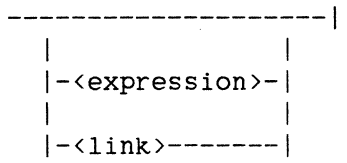
<direction>



<structure>



<condition>



Explanation

(empty)

If the direction parameter is empty, the default value is used, and the FORTRAN77LOCK operation transfers the current record in the specified structure to the user work area.

FIRST

Indicates that the FORTRAN77LOCK operation will transfer the first record in the specified structure to the user work area.

NEXT

Indicates that the FORTRAN77LOCK operation will transfer the next record in the specified structure to the user work area.

PRIOR

Indicates that the FORTRAN77LOCK operation will transfer the prior record in the specified structure to the user work area.

LAST

Indicates that the FORTRAN77LOCK operation will transfer the last record in the specified structure to the user work area.

<data set>

Names a data set in the data base that will be accessed at run time.

<index set>

Names an index set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<expression>

A selection expression used during the FORTRAN77LOCK operation to lock records that meet a specified condition.

<link>

Specifies the data base structure or data set item that links the current record to a record in another data set.

Example

The following program fragments show how FORTRAN77LOCK is used to find and lock the first record in a data set named EMP, which satisfies a specified condition.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBLOCK (STR, ITEM, WHERE)
   CHARACTER STR, ITEM, WHERE
   IN LIBRARY DMI (ACTUALNAME = "FORTRAN77LOCK")
END
```

Setting Variables to Initial Values:

```
 CHARACTER DSNAME      *4      /"EMP"/,
*          FIRST      *10      /"FIRST"/,
*          COND       *9      /"EMP-NO=11"/
```

Invoking the Entry Point:

```
RSLT = DBLOCK(FIRST,DSNAME,COND)
IF (RSLT) CALL EXCEPT
```

FORTRAN77DELETE

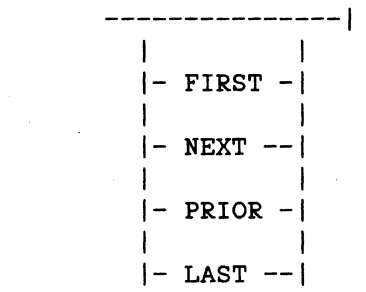
FORTRAN77DELETE deletes a record from the specified data set. The parameters shown below indicate the direction in which the search and deletion operation proceeds, and the selection expression used during the search.

FORTRAN77DELETE functions in the same manner as FORTRAN77FIND, except that a record is deleted after it is found, unless it is a global data record which cannot be deleted. Whenever FORTRAN77DELETE is called, a locking function is implicitly performed to lock the data set record before it is deleted.

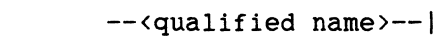
Required Parameters:	Data Type of Parameters:
<direction>	CHARACTER
<data set>	CHARACTER
<condition>	CHARACTER

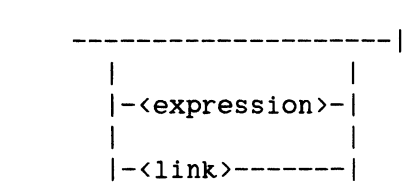
Syntax

<direction>



<data set>





Explanation

(empty)

If the direction parameter is empty, then the FORTRAN77DELETE call will transfer the current record in the specified structure to the user work area before deleting the record.

FIRST

Indicates that the FORTRAN77DELETE call will transfer the first record in the specified structure to the user work area before deleting the record.

NEXT

Indicates that the FORTRAN77DELETE call will transfer the next record in the specified structure to the user work area before deleting the record.

PRIOR

Indicates that the FORTRAN77DELETE call will transfer the prior record in the specified structure to the user work area before deleting the record.

LAST

Indicates that the FORTRAN77DELETE call will transfer the last record in the specified structure to the user work area before deleting the record.

<qualified name>

See the appendix "Syntax Diagrams for Entry Point Parameters" for basic data base constructs that explain the meaning of qualified name in detail.

<expression>

A selection expression used during the FORTRAN77DELETE operation to find and delete records that meet a specified condition.

<link>

Specifies the data base structure or data set item that links the current record to a record in another data set.

Example

The following program fragments show how FORTRAN77DELETE is used to find and delete the first record in a data set named EMP, which satisfies a specified condition.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBDEL (STR, ITEM, WHERE)
   CHARACTER STR, ITEM, WHERE
   IN LIBRARY DMI (ACTUALNAME = "FORTRAN77DELETE")
END
```

Setting Variables to Initial Values:

```
 CHARACTER DSNAME      *4      /"EMP"/,
*          FIRST      *10     /"FIRST"/,
*          COND       *9      /"EMP-NO=11"/
```

Invoking the Entry Point:

```
 RSLT = DBDEL(FIRST,DSNAME,COND)
 IF (RSLT) CALL EXCEPT
```

FORTRAN77VERB

FORTRAN77VERB causes a specified user language interface statement to be performed. The parameters that are used must evaluate to a complete statement. Permissible DMSII user language constructs are shown in the syntax diagrams below, but exception actions and input/output mapping clauses are not allowed. Brief descriptions of these statements are given below, but full descriptions are given in the DMSII User Language Interface Programming Guide.

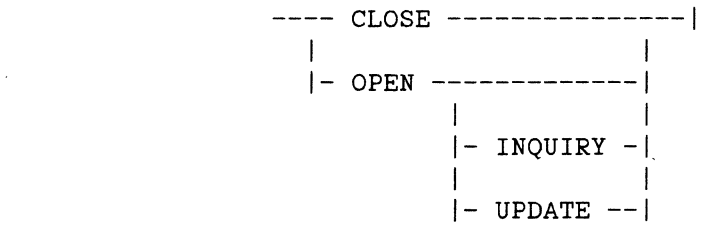
FORTRAN77VERB allows flexible, run-time definition of DMSII operations, and provides users with an easy transition from DMSII user language interface statements to DMINTERPRETER statements. But because it must completely parse its input, FORTRAN77VERB runs at higher processor cost than the other, more specialized entry points.

Required Parameters:

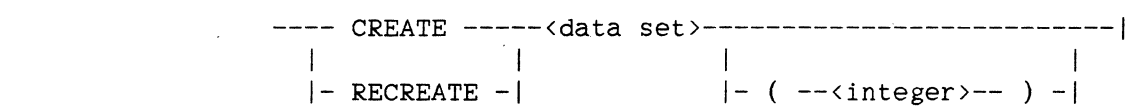
<user language interface statement>

Syntax

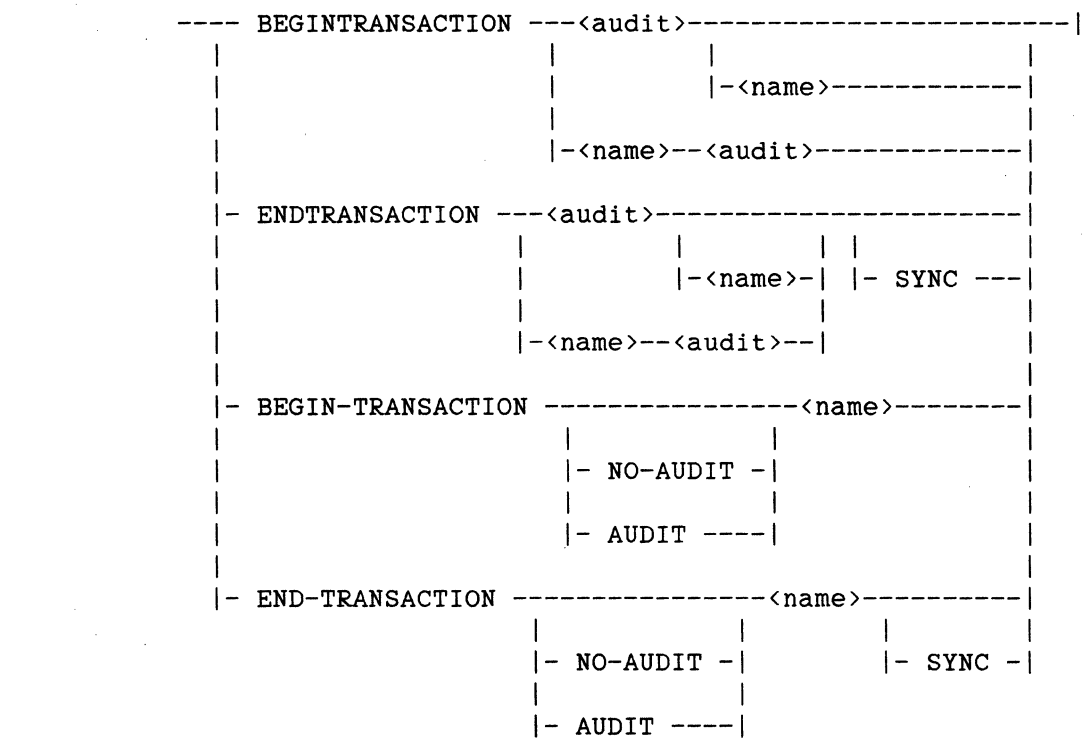
<data base statement>



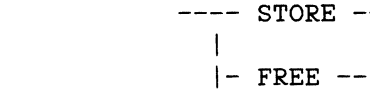
<create statement>



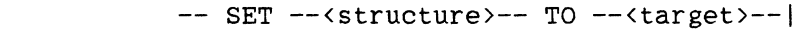
<transaction statement>



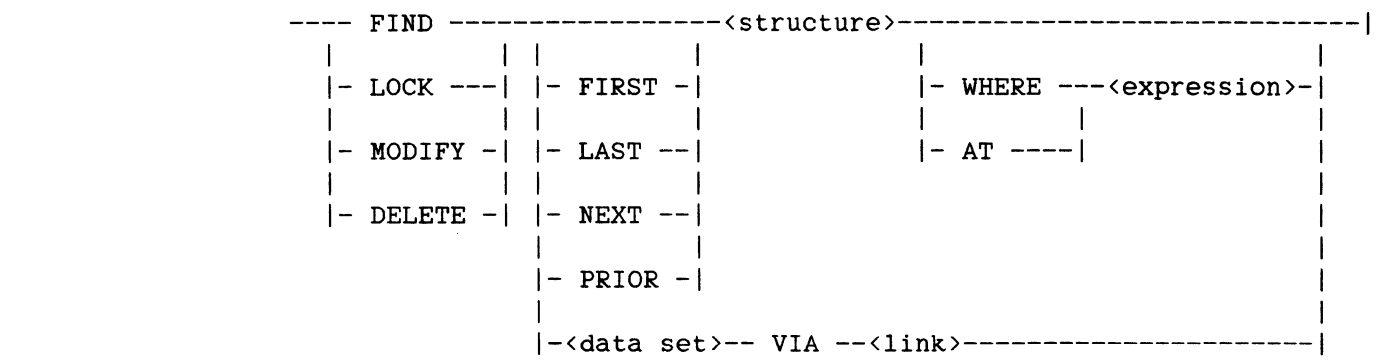
<record statement>



<set statement>



<search statement>



<status statement>

Explanation

<data base statement>

Closes the data base, or opens the data base for inquiry or update.

<create statement>

Initializes the user work area for an existing data set record or a re-created image of a data set record.

<transaction statement>

Causes a program to enter transaction state or exit transaction state, with or without auditing an image of the current restart data set and forcing a syncpoint. The name specifies the restart data set, which is checked for validity when it is present.

<record statement>

Stores a new or modified record in a specified structure, or unlocks a record residing in a specified structure.

<set statement>

Sets a specified structure to its beginning or ending, or to a particular data set if the structure is an index set.

<search statement>

Finds a selected record in a specified structure, and locks the record for modification or deletion.

<status statement>

Allows the user to find out whether a prior data base operation was successful.

Example

The following program fragments show how FORTRAN77VERB can be called to execute a user language interface statement. When using this entry point, parameters for which values have been initialized in the program are not used. Instead, complete user language interface statements must be used. However, when using the status statement construct, just the string "DMSTATUS" is passed as an parameter.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBVERB (STMT)
   CHARACTER STMT
   IN LIBRARY DMI (ACTUALNAME = "FORTRAN77VERB")
END
```

Setting Variables to Initial Values:

Not applicable when using DBVERB.

Invoking the Entry Point:

```
   USING DBVERB ON A SET STATEMENT

RSLT = DBVERB("SET EMP TO BEGINNING")
IF (RSLT) CALL EXCEPT

   USING DBVERB ON A SEARCH STATEMENT

RSLT = DBVERB("FIND FIRST EMP AT EMP-NO=11")
IF (RSLT) CALL EXCEPT

   USING DBVERB ON A STATUS STATEMENT

RSLT = DBVERB("SET EMP TO BEGINNING")
IF (RSLT) CALL EXCEPT
RSLT = DBVERB("FIND PRIOR EMP")
RSLT = DBVERB("DMSTATUS")
CALL DBEXCN(RSLT,MSG)
CALL DBEXCT(RSLT,MSG)
```

6.2 DATA TRANSFER ENTRY POINTS

Data transfer entry points, prefixed with "GET" or "PUT," are provided for moving data to and from user work areas associated with specified data base structures. Each item needed from a data set record must be individually moved to or from a FORTRAN77 variable, by calling an appropriate entry point of the DMINTERPRETER Library, before the item can be manipulated by the program.

The "GET" entry points move data from the data base to variables that have been declared by the programmer. The "PUT" entry points move data from FORTRAN77 expressions to the data base.

The DMINTERPRETER Library checks the parameters of data transfer entry points to ensure that the requested operations are compatible with the Data and Structure Definition Language (DASDL) data base description. For example, items declared in the data base description with an OCCURS clause must be subscripted.

The "PUT" entry points (FORTRAN77PUTCHARACTER, FORTRAN77PUTREAL, FORTRAN77PUTLOGICAL, FORTRAN77PUTDOUBLE, and FORTRAN77PUTNULL) return an exception if the value to be put into the data base does not fit into the specified data item. The "GET" entry points (FORTRAN77GETCHARACTER, FORTRAN77GETREAL, FORTRAN77GETDOUBLE, and FORTRAN77GETLOGICAL) return an exception if the declared variable is too short to hold the specified data item.

FORTRAN77GETCHARACTER

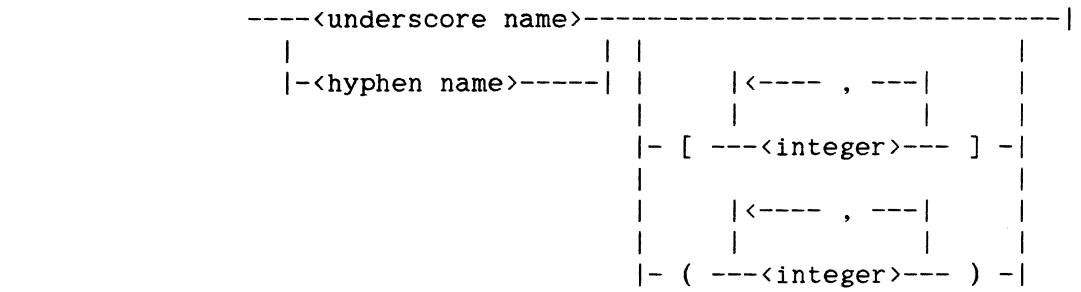
FORTRAN77GETCHARACTER sets a variable declared in the application program to a character string representing the contents of an ALPHA item or GROUP item from the current record in the specified structure.

Required Parameters:	Data Type of Parameters:
<structure>	CHARACTER
<item name>	CHARACTER
<variable>	CHARACTER

Syntax

<structure>

<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

A FORTRAN77 character string variable.

Example

The following program fragments show how FORTRAN77GETCHARACTER is used to move a data item value from the user work area to a variable declared by the programmer. In this example, a value for DSNAME has been initialized at the beginning of the program, whereas the value for the ITMNAM parameter is assigned in the body of the program.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBGETC (STR, ITEM, VAR)
   CHARACTER STR, ITEM, VAR
IN LIBRARY DMI (ACTUALNAME = "FORTRAN77GETCHARACTER")
END
```

Setting Variables to Initial Values:

```
 CHARACTER DSNAME      *4      /"EMP"/,
*          ITMNAM      *13     ,
*          EXPR3       *22
```

Invoking the Entry Point:

```
ITMNAM = "EMP-FNAME"
RSLT = DBGETC(DSNAME,ITMNAM,EXPR3)
IF (RSLT) CALL EXCEPT
```

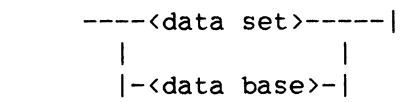
FORTRAN77GETKANJI

FORTRAN77GETKANJI sets a variable declared in the application program to Kanji character string representing the contents of an ALPHA item of which USAGE IS KANJI from the current record in the specified structure.

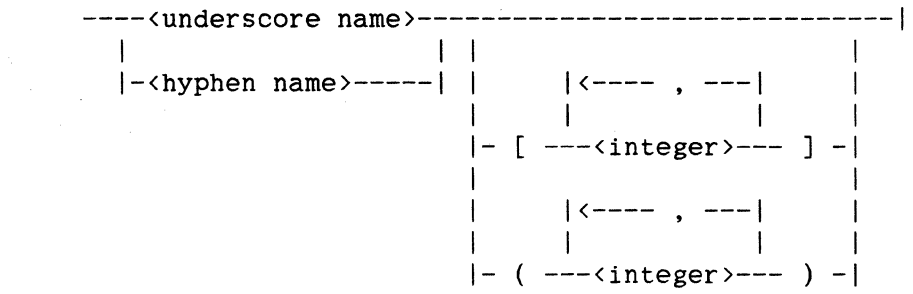
Required Parameters:	Data Type of Parameters:
<structure>	CHARACTER
<item name>	CHARACTER
<variable>	CHARACTER

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

A FORTRAN77 character string variable.

Example

The following program fragments show how FORTRAN77GETKANJI is used to move a data item value from the user work area to a variable declared by the programmer. In this example, a value for DSNAME has been initialized at the beginning of the program, whereas the value for the ITMNAM parameter is assigned in the body of the program.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBGETK (STR, ITEM, VAR)
IN LIBRARY DMI (ACTUALNAME = "FORTRAN77GETKANJI");
END
```

Setting Variable to Initial Values:

```
 CHARACTER DSNAME       *3       /"EMP"/,
*          ITMNAM       *13      ,
*          EXPR3        *11
```

Invoking the Entry Point:

```
ITMNAM = "EMP-KNAME";
RSLT = DBGET(DSNAME,ITMNAM,EXPR3) ;
IF (RSLT) CALL EXCEPT
```

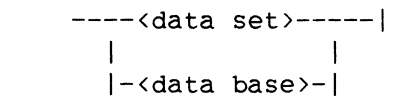
FORTRAN77GETREAL

FORTRAN77GETREAL sets a variable declared in the program to the numeric value of a NUMBER, REAL, FIELD, COUNT, RECORDTYPE, or POPULATION data item in the current record of the specified data base structure. If an exception occurs, the variable is set to HIGH-VALUE (all bits ON), and an appropriate exception is returned. Numeric items that are too large to be retrieved by FORTRAN77GETREAL can be retrieved using the FORTRAN77GETDOUBLE entry point.

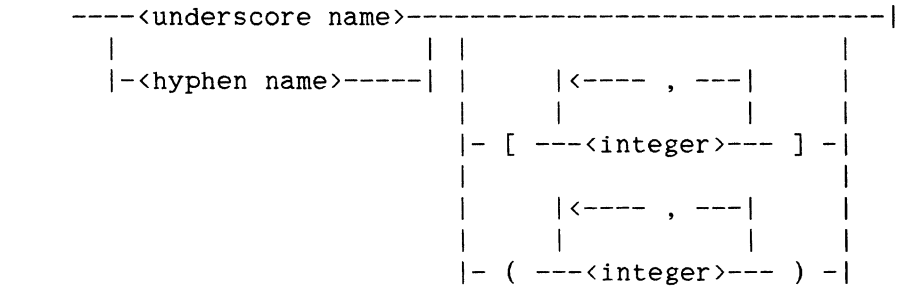
Required Parameters:	Data Type of Parameters:
<structure>	CHARACTER
<item name>	CHARACTER
<variable>	REAL

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

A FORTRAN77 variable that can hold a REAL value.

Example

The following program fragments show how FORTRAN77GETREAL is used to move a data item value from a DMSII user work area to a variable declared by the programmer. In this example, a value for DSNAME has been initialized at the beginning of the program, whereas the value for the ITMNAM parameter is assigned in the body of the program.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBGETR (STR, ITEM, VAR)
   CHARACTER STR, ITEM
   REAL VAR
   IN LIBRARY DMI (ACTUALNAME = "FORTRAN77GETREAL")
END
```

Setting Variables to Initial Values:

```
CHARACTER DSNAME      *4      /"EMP"/,
*         ITMNAM      *13

REAL      EXPR1
```

Invoking the Entry Point:

```
ITMNAM = "EMP-NO"
RSLT = DBGETR(DSNAME,ITMNAM,EXPR1)
IF (RSLT) CALL EXCEPT
```

FORTRAN77GETLOGICAL

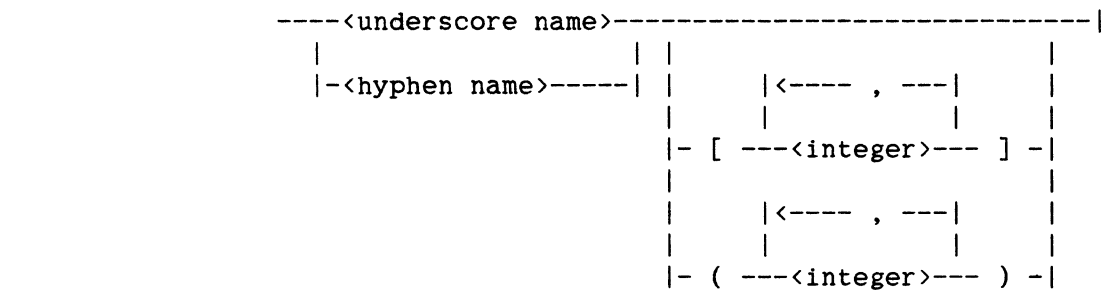
FORTRAN77GETLOGICAL retrieves a BOOLEAN value from a BOOLEAN item or a bit in a FIELD item, and sets a FORTRAN77 logical variable to either ".TRUE." or ".FALSE." depending on the value. The logical variable will be set to a false indication if an exception occurs. The character string that is returned will be truncated if the declared variable is too short.

Required Parameters:	Data Type of Parameters:
<structure>	CHARACTER
<item name>	CHARACTER
<variable>	LOGICAL

Syntax

<structure>

<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

A FORTRAN77 variable declared as LOGICAL.

Example

The following program fragments show how FORTRAN77GETLOGICAL is used to move a data item value from the user work area to a variable declared by the programmer. In this example, a value for DSNAME has been initialized at the beginning of the program, whereas a value for the ITMNAM parameter is assigned in the body of the program.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBGETL (STR, ITEM, VAR)
   CHARACTER STR, ITEM
   LOGICAL VAR
   IN LIBRARY DMI (ACTUALNAME = "FORTRAN77GETLOGICAL")
END
```

Setting Variables to Initial Values:

```
      CHARACTER DSNAME      *4      /"EMP"/,
     *          ITMNAM      *13

      LOGICAL   EXPR4
```

Invoking the Entry Point:

```
ITMNAM = "EMP-SEC-CLEAR"
RSLT = DBGETL(DSNAME,ITMNAM,EXPR4)
IF (RSLT) CALL EXCEPT
```

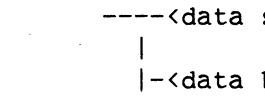
FORTRAN77GETDOUBLE

FORTRAN77GETDOUBLE retrieves numeric data items whose number of significant digits exceeds the capacity of a real variable. FORTRAN77GETDOUBLE sets a variable declared in the program to the numeric value of a NUMBER, REAL, FIELD, COUNT, RECORDTYPE, or POPULATION data item in the current record of the specified structure. If an exception occurs, the declared variable is set to HIGH-VALUE (all BITS on), and an appropriate exception is returned.

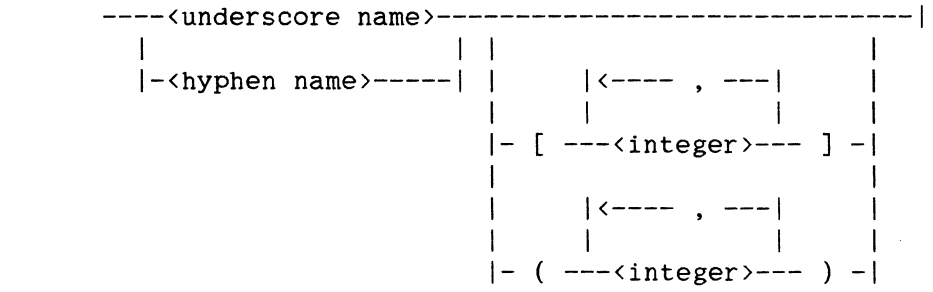
Required Parameters:	Data Type of Parameters:
<structure>	CHARACTER
<item name>	CHARACTER
<variable>	DOUBLE

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

<variable>

A FORTRAN77 double precision variable.

Example

The following program fragments show how FORTRAN77GETDOUBLE is used to move a data item value from a DMSII user work area to a variable declared by the programmer. In this example, a value for DSNAME has been initialized at the beginning of the program, whereas the value for the ITMNAM parameter is assigned in the body of the program.

Library Declaration of Entry Point:

```
      LOGICAL FUNCTION DBGETD (STR, ITEM, VAR)
         CHARACTER STR, ITEM
         DOUBLE PRECISION VAR
         IN LIBRARY DMI (ACTUALNAME = "FORTRAN77GETDOUBLE")
      END
```

Setting Variables to Initial Values:

```
      CHARACTER DSNAME      *4      /"EMP"/,
     *          ITMNAM      *13

      DOUBLE PRECISION  EXPR2
```

Invoking the Entry Point:

```
ITMNAM = "EMP-SSN"
RSLT = DBGETD(DSNAME,ITMNAM,EXPR2)
IF (RSLT) CALL EXCEPT
```

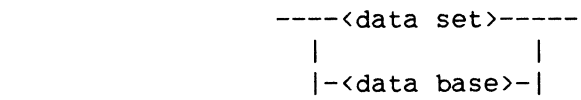
FORTRAN77PUTCHARACTER

FORTRAN77PUTCHARACTER puts a character string into a particular data item in the current record of the specified structure. The data item must be declared in DASDL as an ALPHA or GROUP item. The character string must not exceed the length of the data item where it will be stored, otherwise an exception is returned.

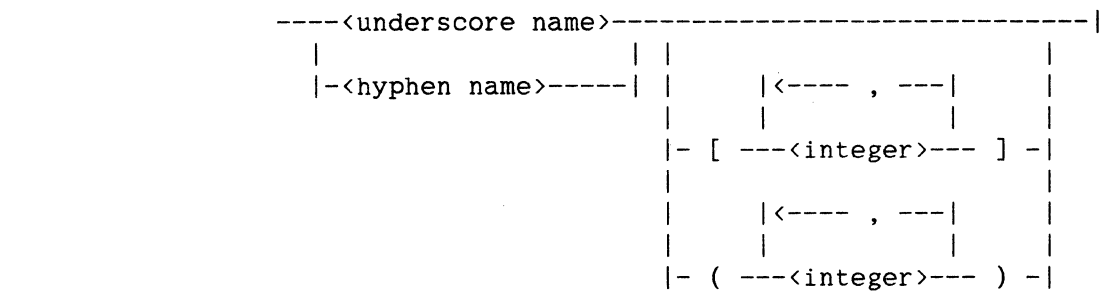
Required Parameters:	Data Type of Parameters:
<structure>	CHARACTER
<item name>	CHARACTER
CHARACTER EXPRESSION	CHARACTER

Syntax

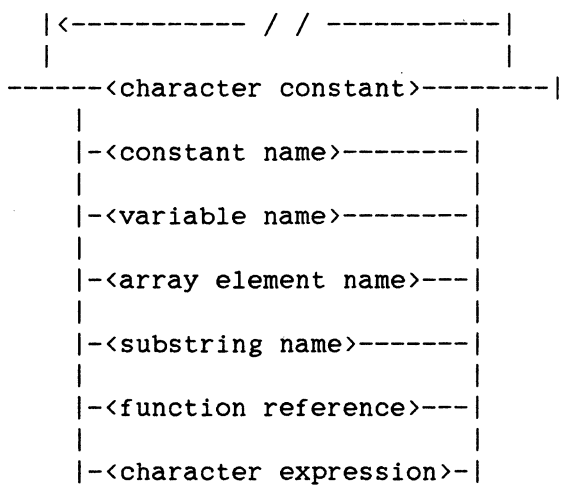
<structure>



<item name>



CHARACTER EXPRESSION



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

CHARACTER EXPRESSION

A FORTRAN77 expression that evaluates to a character string to be put into the data base. For more information, refer to the FORTRAN77 Programming Reference Manual.

Example

The following program fragments show how FORTRAN77PUTCHARACTER is used to move a data item value from a character expression declared by the programmer to a DMSII user work area. In this example, a value for DSNAME has been initialized at the beginning of the program, whereas the values for the ITMNAM and EXPR3 parameters are assigned in the body of the program. Concatenation has also been used to create a string for the employee name being put into the data base.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBPUTC (STR, ITEM, VAR)
   CHARACTER STR, ITEM, VAR
   IN LIBRARY DMI (ACTUALNAME =
    "FORTRAN77PUTCHARACTER")
   END
```

Setting Variables to Initial Values:

```
 CHARACTER DSNAME      *4      /"EMP"/,
*          ITMNAM      *13     ,
*          EXPR3       *22
```

Invoking the Entry Point:

```
ITMNAM = "EMP-FNAME"
EXPR3 = "CATHY"
RSLT = DBPUTC(DSNAME,ITMNAM,EXPR3(1:5) // "LINA")
IF (RSLT) CALL EXCEPT
```

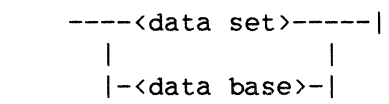
FORTRAN77PUTKANJI

FORTRAN77PUTKANJI puts a character string into a particular data item in the current record of the specified structure. The data item must be declared in DASDL as an ALPHA item of which USAGE IS KANJI. The character string must not exceed the length of the data item where it will be stored; otherwise, an exception is returned.

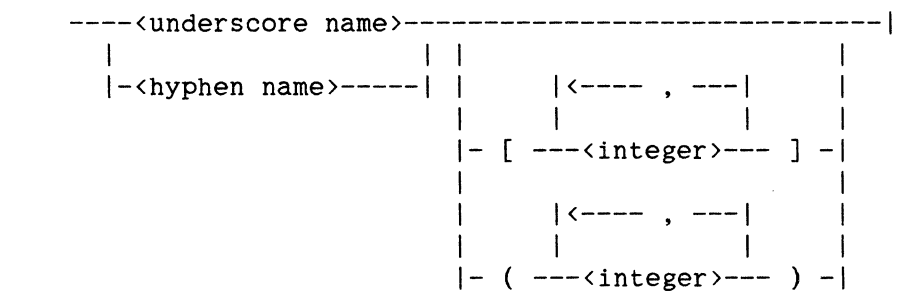
Required Parameters:	Data Type of Parameters:
<structure>	CHARACTER
<item name>	CHARACTER
CHARACTER EXPRESSION	CHARACTER

Syntax

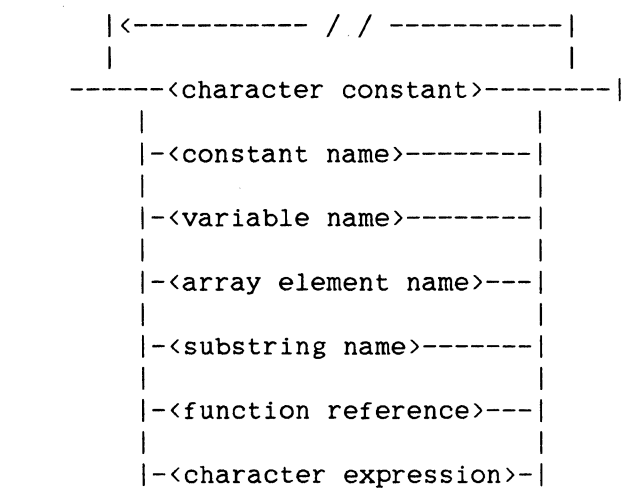
<structure>



<item name>



CHARACTER EXPRESSION



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a variable. Subscripts must be used for OCCURS clause items.

CHARACTER EXPRESSION

A FORTRAN77 expression that evaluates to a character string to be put into the data base. For more information refer to the FORTRAN77 Programming Reference Manual.

Example

The following program fragment shows how FORTRAN77PUTKANJI is used to move a data item value from a character expression declared by the programmer to a DMSII user work area. In this example, a value for DSNAME has been initialized at the beginning of the program, whereas the values for the ITMNAM and EXPR3 parameters are assigned in the body of the program. Concatenation has also been used to create a string for the employee name being put into the data base. KANJI_STRING is a string item that contains Kanji characters without the start-of-Kanji/end-of-Kanji (SOK/EOK) characters.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBPUTK (STR, ITEM, VAR)
   CHARACTER STR, ITEM, VAR
   IN LIBRARY DMI (ACTUALNAME = "FORTRAN77PUTKANJI");
   END
```

Setting Variable to Initial Values:

```
 CHARACTER DSNAME      *3        /"EMP"/,
*          ITMNAM      *13       ,
*          EXPR3       *11
```

Invoking the Entry Point:

```
ITMNAM = "EMPKNAME";
EXPR3 = KNJSTR;
RSLT = DBPUTK(DSNAME,ITMNAM,EXPR3) ;
IF (RSLT) CALL EXCEPT
```

FORTRAN77PUTREAL

FORTRAN77PUTREAL puts a real value into a particular data item in the current record of the specified structure. The item must have been declared in the DASDL as a NUMBER, REAL, or FIELD item. If the parameter for the real value produces an incorrect new value, an exception is returned.

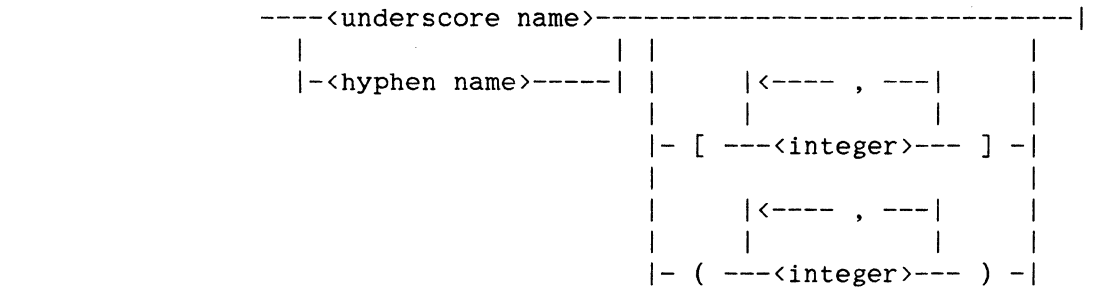
If the number of significant digits in an item exceeds the capacity of a FORTRAN77 arithmetic expression, a FORTRAN77 program should call the FORTRAN77PUTDOUBLE entry point to put real values into items of this type.

Required Parameters:	Data Type of Parameters:
<structure>	CHARACTER
<item name>	CHARACTER
<arithmetic expression>	REAL

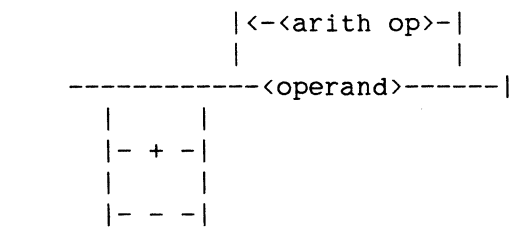
Syntax

<structure>

<item name>



<arithmetic expression>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

<arithmetic expression>

A FORTRAN77 expression that evaluates to a numeric value to be put into the data base. For more information, refer to the FORTRAN77 Programming Reference Manual.

Example

The following program fragments show how FORTRAN77PUTREAL is used to move a data item value from an arithmetic expression declared by the programmer to a DMSII user work area. In this example, a value for DSNAME has been initialized at the beginning of the program, whereas the values for the ITMNAM and EXPR1 parameters are assigned in the body of the program.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBPUTR (STR, ITEM, VAR)
   CHARACTER STR, ITEM
   REAL VAR
   IN LIBRARY DMI (ACTUALNAME = "FORTRAN77PUTREAL")
END
```

Setting Variables to Initial Values:

```
CHARACTER DSNAME      *4      /"EMP"/,
*         ITMNAM      *13

REAL      EXPR1
```

Invoking the Entry Point:

```
ITMNAM = "EMP-NO"
EXPR1 = 11
RSLT = DBPUTR(DSNAME,ITMNAM,EXPR1)
IF (RSLT) CALL EXCEPT
```

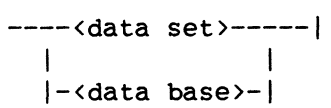
FORTRAN77PUTDOUBLE

FORTRAN77PUTDOUBLE puts a double-precision real value into a data item in the current record of the specified structure. The data item must be a NUMBER, REAL, or FIELD item. If the arithmetic expression produces an incorrect new value, an exception is returned.

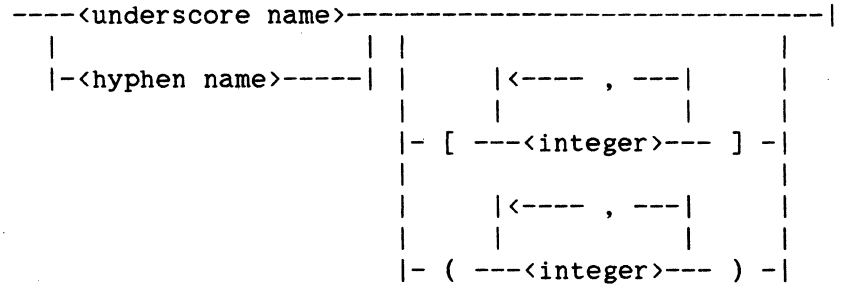
Required Parameters:	Data Type of Parameters:
<structure>	CHARACTER
<item name>	CHARACTER
<arithmetic expression>	DOUBLE

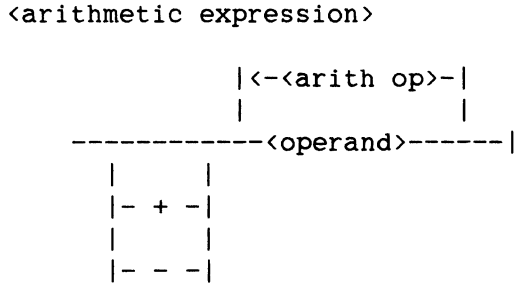
Syntax

<structure>



<item name>





Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

<arithmetic expression>

A FORTRAN77 expression that evaluates to a double-precision numeric value. For more information, refer to the FORTRAN77 Programming Reference Manual.

Example

The following program fragments show how FORTRAN77PUTDOUBLE is used to move a data item value from a real expression declared by the programmer to a DMSII user work area. In this example, a value for DSNAME has been initialized at the beginning of the program, whereas the values for ITMNAM and EXPR2 are assigned in the body of the program. The arithmetic expression used is (EXPR2 - 11).

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBPUTD (STR, ITEM, VAR)
   CHARACTER STR, ITEM
   DOUBLE PRECISION VAR
   IN LIBRARY DMI (ACTUALNAME =
    "FORTRAN77PUTDOUBLE")
END
```

Setting Variables to Initial Values:

```
 CHARACTER DSNAME      *4      /"EMP"/,
*          ITMNAM      *13

 DOUBLE PRECISION      EXPR2
```

Invoking the Entry Point:

```
ITMNAM = "EMP-SSN"
EXPR2 = 573230911
RSLT = DBPUTD(DSNAME,ITMNAM, EXPR2 - 11)
IF (RSLT) CALL EXCEPT
```

FORTRAN77PUTLOGICAL

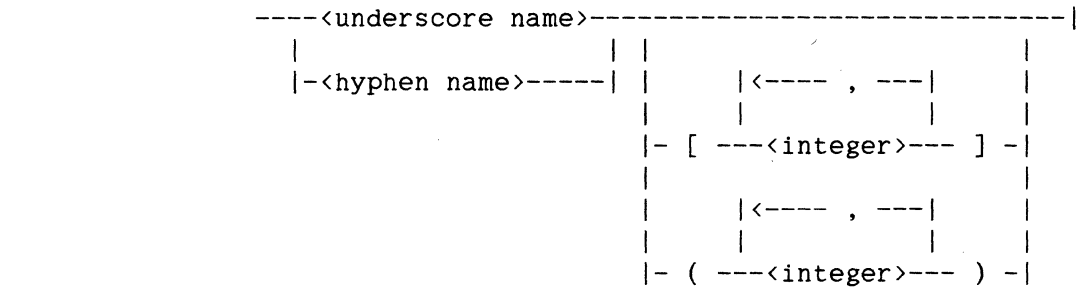
FORTRAN77PUTLOGICAL puts a Boolean value into a particular Boolean data item. The parameter used to carry the Boolean value must be a FORTRAN77 logical expression.

Required Parameters:	Data Type of Parameters:
<structure>	CHARACTER
<item name>	CHARACTER
LOGICAL EXPRESSION	LOGICAL

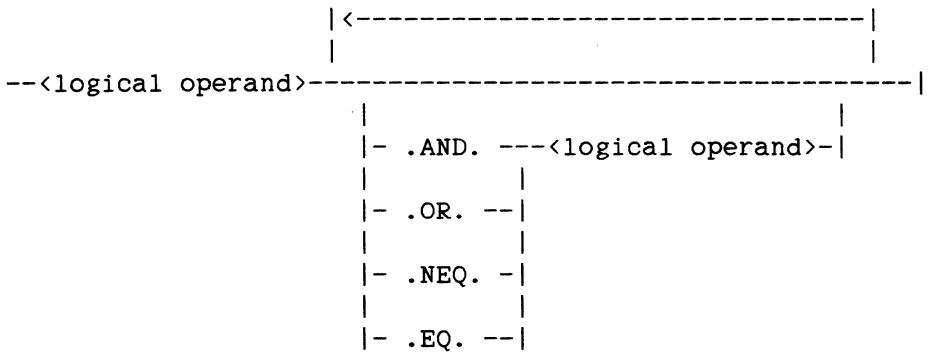
Syntax

<structure>

<item name>



LOGICAL EXPRESSION



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

LOGICAL EXPRESSION

A FORTRAN77 expression that evaluates to ".TRUE." or ".FALSE.". For more information, refer to the FORTRAN77 Programming Reference Manual.

Example

The following program fragments show how FORTRAN77PUTLOGICAL is used to move a logical expression evaluating to ".FALSE." from a FORTRAN77 program to a DMSII user work area. In this example, a value for DSNAME has been initialized at the beginning of the program, whereas the values for ITMNAM and EXPR4 are assigned in the body of the program.

Library Declaration of Entry Point:

```
      LOGICAL FUNCTION DBPUTL (STR, ITEM, VAR)
         CHARACTER STR, ITEM
         LOGICAL VAR
         IN LIBRARY DMI (ACTUALNAME =
          "FORTRAN77PUTLOGICAL")
      END
```

Setting Variables to Initial Values:

```
      CHARACTER DSNAME      *4      /"EMP"/,
     *          ITMNAM      *13

      LOGICAL   EXPR4
```

Invoking the Entry Point:

```
      ITMNAM = "EMP-SEC-CLEAR"
      EXPR4 = .TRUE.
      RSLT = DBPUTL(DSNAME,ITMNAM,.NOT. EXPR4)
      IF (RSLT) CALL EXCEPT
```

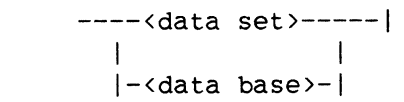
FORTRAN77PUTNULL

FORTRAN77PUTNULL sets a particular data item in the current record of the specified structure to its NULL value. The data item chosen should not be of type LOGICAL, COUNT, RECORDTYPE, or POPULATION.

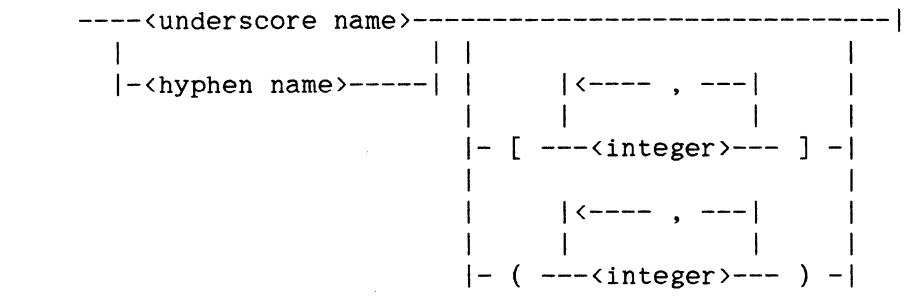
Required Parameters:	Data Type of Parameters:
<structure>	CHARACTER
<item name>	CHARACTER

Syntax

<structure>



<item name>



Explanation

<data set>

Names a data set in the data base that will be accessed at run time.

<data base>

Names the data base for which the DMINTERPRETER Library has been compiled.

<item name>

Specifies the name of the data set item that will be moved to a user work area. Subscripts must be used for OCCURS clause items.

Example

The following program fragments show how FORTRAN77PUTNULL is used to set a data base item to its NULL value.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBPUTN (STR, ITEM)
   CHARACTER STR, ITEM
   IN LIBRARY DMI (ACTUALNAME = "FORTRAN77PUTNULL")
END
```

Setting Variables to Initial Values:

```
CHARACTER DSNAME      *4      /"EMP"/,
*         ITMNAM      *13
```

Invoking the Entry Point:

```
ITMNAM = "EMP-SALARY"
RSLT = DBPUTN(DSNAME,ITMNAM)
IF (RSLT) CALL EXCEPT
```

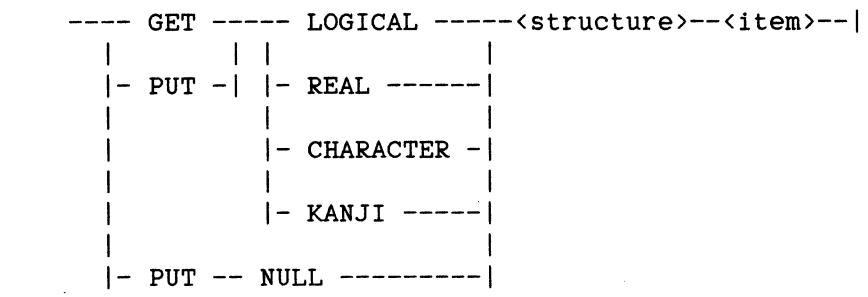
FORTRAN77DATA

FORTRAN77DATA allows "GET" and "PUT" commands to be dynamically constructed during program execution. The data request parameter contains a "GET" or "PUT" operation. The data parameter receives the value for "GET" operations and is used to hold the new value for "PUT" operations.

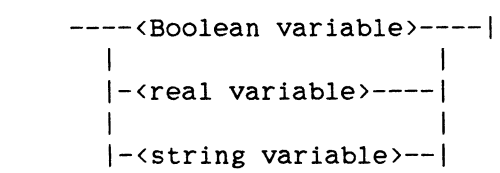
Required Parameters:	Data Type of Parameters:
<data request>	CHARACTER
<data>	CHARACTER

Syntax

<data request>



<data>



Explanation

GET <data type> <structure> <item>

Specifies that a data value of type LOGICAL, REAL, CHARACTER, or KANJI will be moved from an item in a specified data base structure to a variable declared by the programmer.

PUT <data type> <structure> <item>

Specifies that a data value of type LOGICAL, REAL, CHARACTER, KANJI, or NULL will be moved from a variable declared by the programmer to an item in a specified data base structure.

<data>

For PUT operations, this parameter holds data that will be moved to the data base. For GET operations, this parameter receives data from the data base.

Example

The following program fragments show how FORTRAN77DATA is used to move data from an FORTRAN77 program to the data base. The type of operation being used, the type of item being moved, the data set name, the item name, and the data itself are all specified within the body of the program.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBDATA (REQUST, DATUM)
   CHARACTER REQUST, DATUM
   IN LIBRARY DMI (ACTUALNAME = "FORTRAN77DATA")
END
```

Setting Variables to Initial Values:

```
*          DATREQ      *40     ,
*          DATBUF      *22
```

Invoking the Entry Point:

```
DATREQ = "GET REAL EMP EMP-SSN"
RSLT = DBDATA(DATREQ,DATBUF)
IF (RSLT) CALL EXCEPT

DATREQ = "GET REAL EMP EMP-DATE-HIRED"
RSLT = DBDATA(DATREQ,DATBUF)
IF (RSLT) CALL EXCEPT

DATREQ = "GET STRING EMP EMP-LNAME"
RSLT = DBDATA(DATREQ,DATBUF)
IF (RSLT) CALL EXCEPT
```

6.3 EXCEPTION-HANDLING ENTRY POINTS

The exception-handling entry points are among the functions of the interpretive interface that are allowed even when the data base is closed.

All FORTRAN77 entry points of the DMINTERPRETER Library except FORTRAN77EXCEPTIONTEXT and FORTRAN77EXCEPTIONNAME return a particular value to indicate that an exception has occurred. After a FORTRAN77 entry point call to the DMINTERPRETER Library has been completed, the entry point returns an integer to the logical result descriptor in the FORTRAN77 invocation procedure. The returned value is 1 if an exception has occurred and 0 if no exception has occurred. A FORTRAN77 exception is the standard DMSII exception word which includes the exception category and subcategory numbers and the number of the structure involved.

The FORTRAN77STATUS entry point can be subsequently called to retrieve the actual exception, which is the same exception as that returned by the last entry point called. The exception result returned by FORTRAN77STATUS can be passed to FORTRAN77EXCEPTIONNAME and/or FORTRAN77EXCEPTIONTEXT to find out the name of the exception or the full exception message, respectively.

When the FORTRAN77EXCEPTIONTEXT or FORTRAN77EXCEPTIONNAME entry point is called, the DMINTERPRETER Library calls the DMSUPPORT Library to retrieve the text or the name of the exception message. The FORTRAN77EXCEPTIONTEXT or FORTRAN77EXCEPTIONNAME entry points can return any exception that can be returned by the DMSII user language interface. In addition, the DMSUPPORT Library contains a special category of exceptions called INTLIBERRORS, for errors peculiar to the interpretive interface. (Refer to the appendix "DMSII Exceptions and Errors" for a list of exception categories and subcategories and the corresponding exception messages.)

FORTRAN77STATUS

FORTRAN77STATUS returns the exception result of the last entry point call that was made. The program does not need to directly examine the contents of of this exception result, but should pass the result on to FORTRAN77EXCEPTIONNAME or FORTRAN77EXCEPTIONTEXT, which look up the exception message or exception name found in the DMSUPPORT Library.

Required Parameters:	Data Type of Parameters:
None for FORTRAN77	Not applicable

Example

The following program fragments show how FORTRAN77STATUS can be used to obtain an exception result from the last entry point that was called.

Library Declaration of Entry Point:

```
LOGICAL FUNCTION DBSTAT ()
   IN LIBRARY DMI (ACTUALNAME = "FORTRAN77STATUS")
END
```

Setting Variables to Initial Values:

```
LOGICAL    RESULT
```

Invoking the Entry Point:

```
RESULT = DBSTAT
```

FORTRAN77EXCEPTIONTEXT

A call on FORTRAN77EXCEPTIONTEXT should be preceded by a call on FORTRAN77STATUS. After FORTRAN77STATUS returns the exception-word for the latest call on the DMINTERPRETER Library, FORTRAN77EXCEPTIONTEXT can return the text of the exception message in its text parameter.

The exception message can later be displayed by the program. The text parameter contains "NO EXCEPTION" if no exception has occurred. If the text is not long enough to hold the entire message, the message is truncated to fit in the available space. If the space is longer than the message text, then it is padded with trailing blanks.

If an error message is encountered when FORTRAN77EXCEPTIONTEXT is called, the user should investigate whether the DMSUPPORT Library is currently unavailable, possibly for security reasons.

Required Parameters:	Data Type of Parameters:
<exception>	LOGICAL
<text>	CHARACTER

Explanation

<exception>

A logical variable containing the exception information from a DBSTATUS entry point call.

<text>

A character variable with enough room to hold error messages without truncation if possible.

Example

The following procedure shows how FORTRAN77EXCEPTIONTEXT is used to obtain the exception message that corresponds to the exception word returned by the FORTRAN77STATUS entry point.

Library Declaration of Entry Point:

```
      SUBROUTINE DBEXCT (EXCEPT, TEXT)
         LOGICAL EXCEPT
         CHARACTER TEXT
         IN LIBRARY DMI (ACTUALNAME =
                         "FORTRAN77EXCEPTIONTEXT")
      END
```

Exception-Handling Subroutine:

```
      SUBROUTINE EXCEPT(RESULT)

  $ INCLUDE "DATABASE/DMINTERPRETER" 20290000-20299999

      CHARACTER     MSG       *70
      LOGICAL       RESULT

      RESULT = DBSTAT

      CALL DBEXCN(RESULT,MSG)
      WRITE(6,200) MSG

      CALL DBEXCT(RESULT,MSG)
      WRITE(6,200) MSG
  200 FORMAT(1X,A)
      RETURN
      END
```

FORTRAN77EXCEPTIONNAME

A call on FORTRAN77EXCEPTIONNAME should be preceded by a call on FORTRAN77STATUS. For an exception returned by FORTRAN77STATUS, FORTRAN77EXCEPTIONNAME returns in the text parameter the name of the type of exception that occurred. The text parameter contains either the name of a DMSII exception category, such as NOTFOUND or OPENERROR, or "NO EXCEPTION" if no exception has occurred. If the text parameter is not long enough to hold the entire message, the message is truncated to fit in the available space. If the text parameter is longer than the message, the extra space is padded with trailing blanks.

If an error message is encountered when FORTRAN77EXCEPTIONTEXT is called, the user should investigate whether the DMSUPPORT Library is currently unavailable, possibly for security reasons.

Required Parameters:	Data Type of Parameters:
<exception>	LOGICAL
<text>	CHARACTER

Explanation

<exception>

A logical variable containing the exception information from a FORTRAN77STATUS entry point call.

<text>

A character variable with enough room to hold error messages without truncation if possible.

Example

The following procedure shows how FORTRAN77EXCEPTIONNAME is used to obtain the name of the exception that corresponds to the exception result returned by the FORTRAN77STATUS entry point.

Library Declaration of Entry Point:

```
SUBROUTINE DBEXCN (EXCEPT, NAME)
   LOGICAL EXCEPT
   CHARACTER NAME
   IN LIBRARY DMI (ACTUALNAME =
    "FORTRAN77EXCEPTIONNAME")
END
```

Exception-Handling Subroutine:

```
      SUBROUTINE EXCEPT(RESULT)

 $ INCLUDE "DATABASE/DMINTERPRETER" 20290000-20299999

      CHARACTER     MSG       *70
      LOGICAL       RESULT

      RESULT = DBSTAT

      CALL DBEXCN(RESULT,MSG)
      WRITE(6,200) MSG

      CALL DBEXCT(RESULT,MSG)
      WRITE(6,200) MSG
  200 FORMAT(1X,A)
      RETURN
      END
```

6.4 ATTRIBUTE-SETTING ENTRY POINTS

FORTRAN77SETLIMIT, the only FORTRAN77 attribute-setting entry point, is used to restrict the number of times the DMINTERPRETER can call the Accessroutines to complete a find. FINDLIMIT is the only valid limit type allowed for the FORTRAN77SETLIMIT entry point.

FORTRAN77SETLIMIT can be used to declare DMINTERPRETER entry points in application programs, unless renaming is desired or necessary due to limitations of a particular language.

FORTRAN77SETLIMIT

FORTRAN77SETLIMIT sets the various limit values for the DMINTERPRETER.

Required Parameters:	Data Type of Parameters:
<limittype>	CHARACTER
<limitvalue>	CHARACTER

Explanation

<limittype>

A string parameter which specifies the type of limit to be set, and specifies the maximum number of times the DMINTERPRETER can call the Accessroutines to perform a FIND or LOCK.

FINDLIMIT is the only <limittype> currently defined. By setting FINDLIMIT, the user can prevent long linear searches.

If an invalid <limittype> is specified, INTLIBERROR 33 is returned. If the DMINTERPRETER exceeds the FINDLIMIT while performing a FIND or LOCK, INTLIBERROR 35 is returned.

<limitvalue>

An integer string parameter which specifies the value to be assigned to <limittype>. Negative values are not allowed for <limitvalue>. Specifying a <limitvalue> of zero disables the FINDLIMIT mechanism, allowing infinitely long linear searches.

When the DMINTERPRETER is first initialized, FINDLIMIT is set at the default value of zero.

7 DETERMINING THE DATA BASE STRUCTURE

When writing application programs that will access the interpretive interface to Data Management System II (DMSII), the structure of the data base does not have to be known. Instead, the program can determine the data base structure at run time. In addition, it may be desirable to write general-purpose routines that can access any data base and yet be data-independent. The DMINTERPRETER provides an intrinsic, read-only data set named DBSTRUCTURE to allow programs these capabilities.

DBSTRUCTURE holds structural information about the data base. Unlike ordinary data sets, DBSTRUCTURE can be accessed before the data base is opened. However, as in an ordinary data set, the items in DBSTRUCTURE are accessed by calling the DBFIND entry point. For example, calling DBFIND in the following form eventually locates all DBSTRUCTURE entries if DBFIND is put in a loop that terminates only after all required items have been found:

```
DBFIND ("NEXT", "DBSTRUCTURE", "ITEM-REQUIRED")
```

DBSTRUCTURE is generated when the DMINTERPRETER is compiled; it need not be declared in the Data and Structure Definition Language (DASDL) description of the data base. No sets or subsets span DBSTRUCTURE, and the only valid operations that can be used with DBSTRUCTURE are DBFIND, DBGET, and DBSET entry point calls.

Use of the DBSTRUCTURE intrinsic data set is optional. It can be omitted by specifying the SELECTED DATASETS option in BUILDINQ without selecting DBSTRUCTURE. (For more information, refer to "Generation from Remote Terminals" in Section 2 of this manual.) If the data base already contains an actual data set named DBSTRUCTURE, then the intrinsic data set DBSTRUCTURE is not generated. (A user-declared data set named DBSTRUCTURE cannot, of course, be accessed when the data base is closed.)

DBSTRUCTURE behaves as if it had the following DASDL description:

```
DBSTRUCTURE DATA SET

  (DB-ID                      FIELD(16)       REQUIRED;
   DB-OWNER                   FIELD(16)       REQUIRED;
   DB-NAME                    ALPHA(17)       REQUIRED;
   DB-TYPE                    ALPHA(17)       REQUIRED;

   % ITEMS PRESENT WHEN DB-TYPE = "DATASET"

   DATASET-SUBTYPE            ALPHA(17);

   % ITEMS PRESENT WHEN DB-TYPE = "SET"

   SET-SUBTYPE                ALPHA(17);
   SET-ACCESS-METHOD          ALPHA(17);
   SET-SPANS-DATASET          NUMBER(4);

   % ITEMS PRESENT WHEN DB-TYPE = "ITEM"

   ITEM-SUBTYPE               ALPHA(17);
   ITEM-SUBSCRIPTS            NUMBER(2);
   ITEM-OCCURS                NUMBER(4);
   ITEM-OWNER-GROUP           NUMBER(4);
   ITEM-REQUIRED              BOOLEAN;
   ITEM-SIZE                  NUMBER(4);
   ITEM-SCALE-FACTOR          NUMBER(2);
   ITEM-SIGNED                BOOLEAN;
   ITEM-KEY-CLASS             ALPHA(17);
   ITEM-RECORD-TYPE           NUMBER(3);

   % ITEMS PRESENT WHEN DB-TYPE = "LINK"

   LINK-TO-DATASET            NUMBER(4);
   LINK-OCCURS                NUMBER(4);
  );
```

DBSTRUCTURE contains an entry for each data set, set, subset, access, item, and link in the data base. The first four items in the record are prefixed by "DB" and have non-NULL values in every DBSTRUCTURE record. All other items in DBSTRUCTURE are prefixed by a value for DB-TYPE; they have non-NULL values in a given record if the value of DB-TYPE in that record corresponds to their prefix. For example, if DB-TYPE is "LINK," then only the items prefixed by "LINK" have non-NULL values.

Sets that have GROUP keys have two entries in the DBSTRUCTURE intrinsic data set:

1. An entry that lists the group as its key.

2. An entry that lists the elementary items as its keys.

Each data base also has an entry for global data. This entry can be used to find all of the disjoint structures and global data items in the data base.

DBSTRUCTURE ITEMS

The following paragraphs describe the individual items in DBSTRUCTURE.

DB-ID

This item contains a positive number that uniquely identifies each component in the data base. DB-ID allows records in DBSTRUCTURE to be associated with each other. Items in more than one variable record type are considered to be different entities with different DB-IDs.

DB-OWNER

This item contains the DB-ID of the data set that owns this entry. DB-OWNER is 0 (zero) for the global data entry. All disjoint structures and global data items have the global data's DB-ID in this item.

DB-NAME

This item contains the name of the entity padded with trailing blanks.

DB-TYPE

This item contains a string describing the type of the entity.

DATASET
   Indicates a data set.

SET
   Indicates a set, subset, or access.

ITEM
   Indicates a data item.

LINK
   Indicates a link.

DATASET-SUBTYPE

This item contains a string describing the type of a data set. Valid values are STANDARD, RANDOM, ORDERED, UNORDERED, GLOBAL, DIRECT, RESTART, COMPACT, and STRUCTURE (for DBSTRUCTURE itself).

SET-SUBTYPE

This item contains a string describing the type of a set. Valid values are SET, SUBSET, and ACCESS.

SET-SEARCH-METHOD

This item contains a string describing the accessing method used by a set.

DIRECT
Indicates DIRECT data set accesses.

BINARY
Indicates ORDERED data set accesses and INDEX SEQUENTIAL and ORDERED LIST sets.

HASH
Indicates RANDOM data set accesses and INDEX RANDOM sets.

LINEAR
Indicates UNORDERED data set accesses and BIT VECTOR sets.

SET-SPANS-DATASET

This item contains the DB-ID of the data set spanned by a set.

ITEM-SUBTYPE

This item contains a string describing the type of an item. Valid values are ALPHA, GROUP, FIELD, NUMBER, REAL, BOOLEAN, COUNT, TYPE, and POPULATION.

ITEM-SUBSCRIPTS

This item contains the number of subscripts required when using an item. If no subscripts are required, ITEM-SUBSCRIPTS is 0 (zero).

ITEM-OCCURS

This item stores the subscript limit for an item declared in DASDL with an OCCURS clause. If ITEM-SUBSCRIPTS is 0 (zero), the ITEM-OCCURS contains a NULL. To find the other subscript limits for multiple-subscripted items, inspect the ITEM-OCCURS entries where DB-ID equals ITEM-OWNER-GROUP (described below). Items that have a non-zero value in ITEM-SUBSCRIPTS and a NULL in ITEM-OCCURS are nonoccurring items.

ITEM-OWNER-GROUP

This item contains the DB-ID of the next higher group or field that contains an item. The DB-ID can be used to determine the index limit for items in OCCURring groups and fields, or to determine which items belong to a given group or field. If an item does not belong to a group or field, ITEM-OWNER-GROUP is 0.

ITEM-REQUIRED

This item is TRUE if the entity is a REQUIRED item and FALSE otherwise.

ITEM-SIZE

ITEM-SIZE is non-NULL for NUMBER, FIELD, ALPHA, and GROUP items only. This item contains the declared length of an item in hexadecimal digits, bits, EBCDIC bytes, and bytes, respectively. Ordinarily, ITEM-SIZE does not include any spaces for signed values. However, for GROUP items containing signed NUMBER items, ITEM-SIZE does include spaces for all signs.

ITEM-SCALE-FACTOR

This item contains the declared number of digits to the right of the decimal point for NUMBER and REAL items that have a scale factor declared in DASDL. A NUMBER value cannot exceed

```
10**((ITEM-SIZE)-(ITEM-SCALE-FACTOR))
```

A REAL value must be less than

```
549755813887.5/10**(ITEM-SCALE-FACTOR)
```

unless the REAL value was declared without precision.

ITEM-SIGNED

This item is TRUE for a NUMBER or REAL item only if the NUMBER or REAL item can be negative (in other words, declared as signed in DASDL).

ITEM-KEY-CLASS

This item contains a string indicating the the key status of an item in a spanning set.

MAJOR
The item is the major key of a spanning set or access. (The major key is defined as the first key listed in the declaration of the set.)

MINOR
The item is not the major key of a spanning set or access, but it is a key of the set.

DATA
The item only appears as key data in spanning sets or accesses.

NONE
The item does not appear in the key area of any access or spanning set.

ITEM-RECORD-TYPE

This item contains the record type of an item. ITEM-RECORD-TYPE is 0 (zero) for items in fixed records or in the fixed part of variable-format records. Because an item with a particular DB-ID can only be in records of one type, any other item of the same name in a different record type has a different DB-ID.

ITEM-USAGE

This item contains a string describing the usage of an item. Valid values are EBCDIC and KANJI.

LINK-TO-DATASET

This item contains the DB-ID of the data set to which a link points.

LINK-OCCURS

This item contains the limit of the OCCURS clause for a link. If LINK-OCCURS is 0 (zero), no OCCURS clause was present in the declaration of the link.

A UNDERSTANDING RAILROAD DIAGRAMS

WHAT IS A RAILROAD DIAGRAM?

A railroad diagram is a way of representing the syntax of a command or statement graphically. It shows which items are required or optional, the order in which they should appear, how often you can repeat them, and any required punctuation.

HOW TO READ A RAILROAD DIAGRAM

Normally, you read a railroad diagram from left to right. However, there are some exceptions; in those cases, arrows indicate a right-to-left direction.

If a diagram is too long to fit on one line and must continue on the next, a right arrow (>) appears at the end of the first line and another at the beginning of the next line, like this:

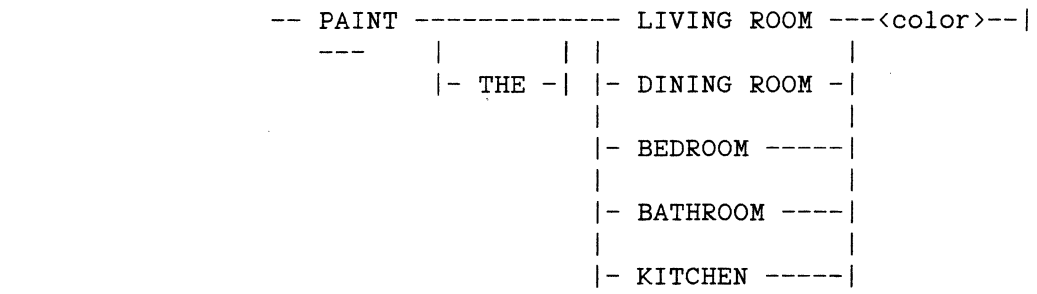
```
-------------------------------------------------------------------------->

>-----------
```

The end of a railroad diagram is denoted by a vertical bar (|) or percent sign (%). The vertical bar means the command or statement can be followed by a semicolon and another command or statement. The percent sign means the command or statement must be on a line by itself.

CONSTANTS AND VARIABLES

Consider a hypothetical command for giving instructions to a house painter:



This command tells the painter to paint a designated room in the color you specify.

The example introduces two important features of railroad diagrams:

- Constants
- Variables

Constants

Constants are items that you cannot vary. You must enter a constant as it appears in the diagram, either in full or abbreviated. If you abbreviate a constant, you must enter everything that is underlined in the railroad diagram, optionally followed by one or more of the remaining characters.

You can recognize constants in railroad diagrams by the fact that they are never enclosed in angle brackets.

In the example, the word PAINT is a constant. You could enter PAINT in full or abbreviate it to PAI or PAIN, but not to PA or PAN. If no part of the constant is underlined, you cannot abbreviate it at all.

Variables

Variables are items that you can replace with other data to suit a particular situation; that is, you can vary the information you enter in place of the variable, subject to rules defined for the particular command or statement.

Variables appear in a railroad diagram enclosed in angle brackets (<>).

In the example, <color> is a variable item. If the description of the PAINT command defines the allowable colors as BLUE, GREEN, PINK, and YELLOW, you can enter any one of these in your command.

FOLLOWING THE PATHS OF A RAILROAD DIAGRAM

The paths of a railroad diagram lead you through the diagram from beginning to end. They are represented by horizontal and vertical lines.

A path shows the allowable syntax. Some diagrams have just one path that goes from the beginning to the end of the diagram. Others contain several paths, each covering a part of the diagram. A path shows which items you can include in a command or statement, which you can omit, and the number of times you can include a particular item or group of items.

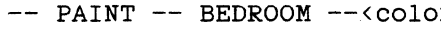
To follow a path through a railroad diagram, you need to understand the items you may encounter along the way. These items are

- Required items and punctuation
- Optional items
- Loops

A description of each item follows.

Required Items and Punctuation

Required items and punctuation must be entered in the command or statement; you cannot omit them. A required item appears by itself in a path (horizontal line). A required item can be either a constant or a variable. For example, if a railroad diagram indicates



the words PAINT and BEDROOM are required constants, and <color> is a required variable. You could correctly enter

```
PAINT BEDROOM BLUE
```

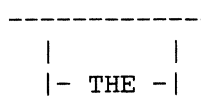
but not

```
PAINT BEDROOM
```

because the required item <color> would be missing.

Optional Items

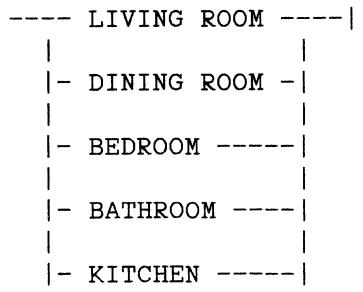
Optional items appear one below another in a vertical list. You can choose any one of the items in the list. If the list also contains an empty path (all dashes), you can omit the item entirely. An optional item can be either a variable or a constant. The PAINT command in the example contains two lists. The first is



which gives you two options:

- Enter the constant THE
- Omit it (because there is an empty path)

The second list has five optional constants:



You must enter one of the optional items (LIVING ROOM, DINING ROOM, BEDROOM, BATHROOM, or KITCHEN) because there is no empty path in this list.

Loops

A loop is an item or group of items that you can repeat. The number of repetitions allowed is controlled by an item called the bridge.

A loop can span all or part of a railroad diagram. It always consists of at least two horizontal lines, one below the other, like this:

```
     |<------- <return character> ---------|
     |                                      |
-----<bridge>--<content of the loop>-------
```

or

```
     |<-- <bridge> -- <return character> --|
     |                                     |
--------- <content of the loop> -----------
```

The bridge shows the maximum number of times you can go through the loop. The bridge can precede the contents of the loop, or it can precede the return character on the upper line of the loop to specify the number of times the right-to-left path can be traversed. The bridge is an integer enclosed in sloping lines, / \, for example, /4\. Not all loops have bridges. Those that do not can be repeated any number of times.

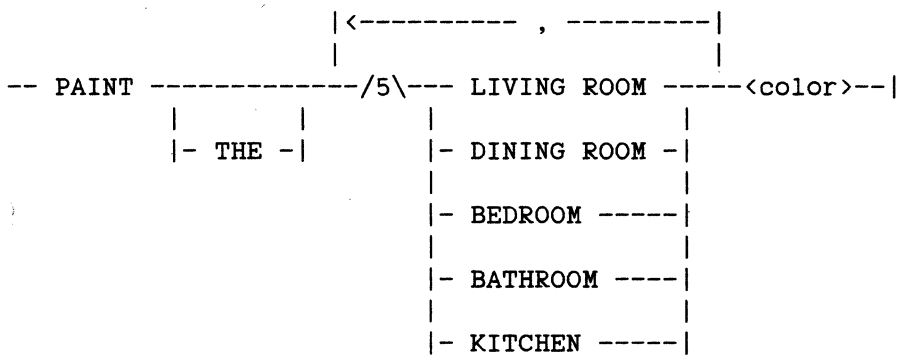
The top line is a right-to-left path that contains information about repeating the loop. The return character is the character to use before each repetition of the loop (often, a comma). Not all loops contain a return character; if none is shown, just enter one or more spaces before repeating the loop.

The other lines show the content of the loop (the data you enter each time you go through the loop). This can be any combination of optional items, required items, lists, and even other loops. The content of a loop can range from simple (one item), to very complex (many items, lists, and loops).

Example 1. A Simple Loop

The PAINT command as first shown is of limited usefulness. To tell the painter to do several rooms, you need a separate command for each room. It would be much easier if you could tell him to do several rooms in one command.

You can do that by making the list of rooms into a loop. The command would then look like this:



The bridge has a value of 5, so you can go through the loop up to five times, for a total of five rooms. The return character is a comma, which you must enter before repeating the loop content.

You can now enter

```
PAINT THE LIVING ROOM, BEDROOM, KITCHEN YELLOW
```

or

```
PAINT DINING ROOM, BEDROOM, BATHROOM BLUE
```

or

```
PAINT BEDROOM PINK
```

or

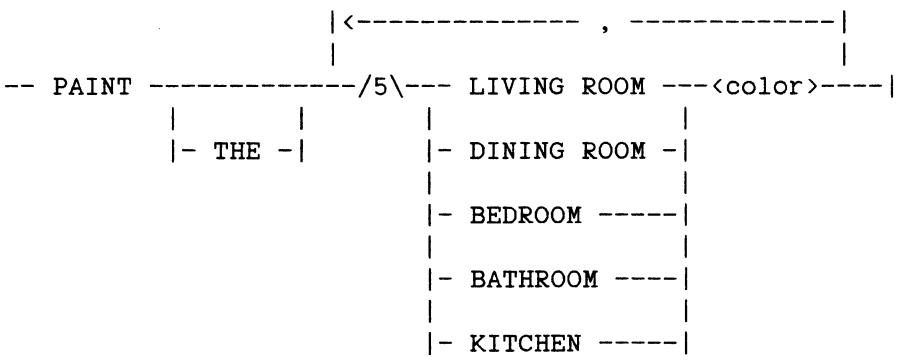
```
PAINT BEDROOM, BATHROOM, BEDROOM, BEDROOM BLUE
```

or any other valid combination.

This simple loop makes the PAINT command more versatile, but a significant drawback remains. Although you can include up to five rooms in a command, you cannot specify different colors.

Example 2. A More Complex Loop

If the content of the loop were to include the color, you could specify a different color for each room.



The content of the loop now consists of the

- List of optional constants that indicate rooms
- Required variable <color>

The bridge value is 5, and the return character is a comma. Given this railroad diagram, some valid PAINT commands would be

PAINT THE BEDROOM PINK

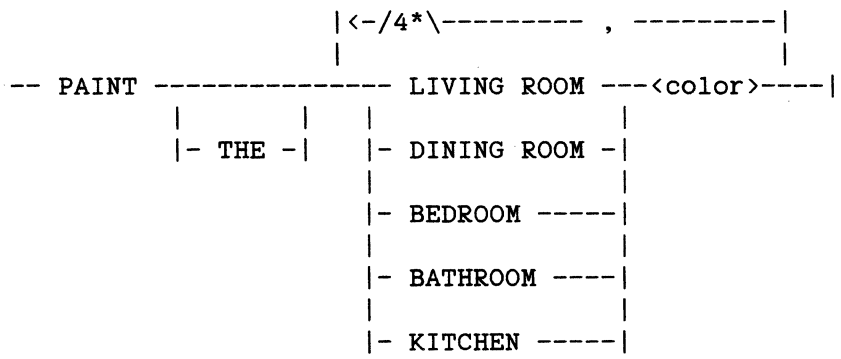
PAINT THE LIVING ROOM BLUE, DINING ROOM GREEN, KITCHEN YELLOW

PAINT BEDROOM GREEN, KITCHEN BLUE

and so on.

Example 3. Another Loop

In some bridges an asterisk follows the number. For example,



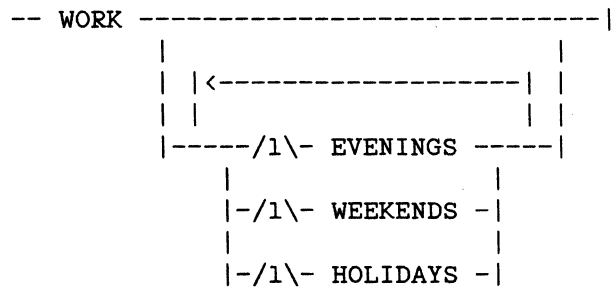
The asterisk means you must take the right-to-left path at least once. You cannot, for example, enter PAINT BEDROOM BLUE; you must tell the painter at least two rooms to paint. The maximum number of rooms to be painted is still five: the first time through the loop with up to four repetitions.

A valid form of the command would be

```
PAINT BEDROOM BLUE, KITCHEN YELLOW
```

Example 4. Another Use of the Bridge

A bridge can also control the number of times you take an individual path within a loop. For example, another command to the painter might be:



Each bridge /1\ indicates you can take that path once or not at all. That is, you can enter each of the items EVENINGS, WEEKENDS, and HOLIDAYS once at most. Some valid commands are

    WORK EVENINGS WEEKENDS HOLIDAYS

    WORK WEEKENDS

    WORK HOLIDAYS EVENINGS

but

    WORK EVENINGS EVENINGS

is invalid.

A FINAL WORD

To familiarize you with railroad diagrams, this explanation describes the elements of the diagrams and gives a few simplified examples. Some of the actual diagrams you will encounter in a book may be considerably more complex.

However, the principles are the same no matter how complex the diagram. The more you work with railroad notation, the more easily you will understand even the most complex diagrams.

B DMSII EXCEPTIONS AND ERRORS

This appendix lists all exceptions and errors that can be returned by the Accessroutines when data management operations are attempted.

An exception or error is returned to an application program whenever a requested data management operation cannot be successfully or correctly completed. Exceptions arise from problems within an application program, and affect only the program that detects the exception situation. Other application programs are unaffected, unless the program receiving the exception causes an abort.

A special category of exceptions called INTLIBERRORs (category number 21) is returned only to programs that are accessing the interpretive interface.

Errors such as system, fatal, integrity, and version errors arise from problems within the data base rather than problems within particular application programs. All programs that request similar data management operations will receive the same errors.

The following chart shows the major category numbers for exceptions and errors, and the mnemonics associated with them. Throughout the rest of this section, subcategory numbers are listed with each mnemonic to indicate which variation of the exception or error has occurred.

Category No.	Mnemonic	Category No.	Mnemonic
1	NOTFOUND	11	OPENERROR
2	DUPLICATES	12	CLOSEERROR
3	DEADLOCK	13	NORECORD
4	DATAERROR	14	INUSE
5	NOTLOCKED	15	AUDITERROR
6	KEYCHANGED	16	ABORT
7	SYSTEMERROR	17	SECURITYERROR
8	READONLY	18	VERSIONERROR
9	IOERROR	19	FATALERROR
10	LIMITERROR	20	INTEGRITYERRO
		21	INTLIBERROR

MNEMONIC/SUBCATEGORY NO./MEANING

ABORT

1 The data base was aborted and rolled back.

2 An abort occurred and the TPS rerun was not finished.

AUDITERROR

1 A BEGINTRANSACTION statement was attempted when the data base was already in transaction state.

2 An ENDTRANSACTION statement was attempted when the data base was not in transaction state.

3 A MIDTRANSACTION statement was attempted when the data base was not in transaction state.

4 A normal BEGINTRANSACTION or ENDTRANSACTION statement was attempted on a data base that was opened by TPS.

5 A TPS BEGINTRANSACTION or ENDTRANSACTION statement was attempted on a data base that was not opened by TPS.

6 The transaction record had an improper TRSTATE value.

7 A MIDTRANSACTION statement was attempted on a data base that was not opened by TPS.

8 The program was not in transaction state when an update operation on an audited data base was attempted.

9 An attempt was made to abort a single transaction while the program was not in transaction state.

10 An attempt was made to write a savepoint record while the program was not in transaction state.

11 An attempt was made either to abort a single transaction or to write a savepoint record, but the DASDL option INDEPENDENTTRANS was not set.

CLOSEERROR

1 The data base declaration was not open.

2 An Accessroutines error occurred on one or more structures.

DATAERROR

1 Some required items were NULL or the VERIFY condition failed.

2 An attempt was made to create variable-format records with an invalid value for the record type.

3 The partition variable yielded an illegal file identifier.

4 READONLY items were changed and now cannot be stored.

DEADLOCK

1 A deadly embrace occurred during an attempt to lock a record.

NOTE

The system will have automatically freed all records for this program.

2 A time-out occurred during an attempt to lock a record.

3 LOCKLIMIT exceeded. Too many records in this structure are currently locked.

4 This error message is no longer in use.

DUPLICATES

1 A STORE statement was attempted and duplicates were not allowed in the index set.

2 An INSERT statement was attempted and duplicates were not allowed in the subset or the data set record was already in the subset.

FATALERROR

(No exception subcategories are returned for large systems.)

INTEGRITYERROR

1 Key and data did not match.

2 An attempt was made to store a record when an automatic set or subset entry was missing.

3 An attempt was made to delete a record when an automatic set or subset entry was missing.

4 An automatic set or subset was pointing to a deleted record.

5 An attempt was made to allocate more than 1000 rows in a file.

INTLIBERROR

1 The first token of the parameter to DBVERB or DBDATA was not a valid verb.

2 The specified string parameter was not a valid key word.

3 The structure name parameter was a data base ID, but not the correct structure type.

4 The structure or link name was not a known data base ID.

5 Conflicting or insufficient qualification was given for the structure or link name.

6 The item name was not a data base identifier.

7 The item type did not correspond to the requested data type.

8 The specified item was not in the specified structure.

9 An invalid index was given for an OCCURring item.

10 An incorrect number of subscripts was given for an OCCURring item.

11 The given item was not in the current variable record type, or incorrect record type-syntax was given.

12 A negative value was placed in an unsigned item.

13 A "get" for this item found that the value was NULL.

14 The string parameter to PUTSTRING was not the same length as the item.

15 The structure or link name was invalid.

16 The item name was invalid.

17 An invalid literal appeared in a selection expression.

18 A right parenthesis was missing in a selection expression.

19 A relational operator was expected in a selection expression.

20 A relation specified a NULL for a REQUIRED item, or the operator was not "=" or "^=".

21 The selection expression did not end correctly, or some unknown error occurred.

22 A numeric argument to a "put" was invalid for the item.

23 This operation was not valid for global data.

24 An item and a literal had mismatched length or type in a selection expression.

25 More than 47 relations appeared in a selection expression.

26 No valid set spanned the embedded data set.

27 A software failure occurred in a selection expression tree execution.

28 An invalid request was made while the data base was closed.

29 A store or delete operation was attempted while the program was not in transaction state.

30 An index set was left undefined by a data set condition search.

31 The parameter was too short to hold the response.

32 The value of a NUMBER item did not fit into a REAL parameter.

33 The Limittype parameter was invalid.

34 The Limitvalue parameter was invalid.

35 The FINDLIMIT value was exceeded.

INUSE

1 An attempt was made to delete a record without setting its references and embedded set to NULL.

2 An attempt was made to delete a record when the count item was not zero.

3 An attempt was made to delete a record with a non-NULL embedded structure.

IOERROR

1 The unit was not ready.

2 A read parity error occurred.

4 A descriptor error occurred.

5 "NO SPACE" condition has been encountered.

This type of exception can occur under the following conditions:

1. A DMSII data file is contained on a multipack family.
2. One of the packs of the multipack family is re-created.
3. A DMSII operation (for example, FIND or LOCK) is attempted on a record that was in an area that was on the newly re-created pack.

6 An attempt was made to read past the end of file.

7 An attempt was made to read or write past the end of the last row.

8 The I/O was canceled.

11 A pack write-lockout occurred.

12 A multiplexor or controller error occurred.

13 The pack drive failed.

14 A pack sector format error occurred.

15 A storage unit time-out occurred.

16 A row was locked out due to a previous write error.

17 The checksum failed.

99 Unknown.

256 The ADDRESSCHECK failed.

KEYCHANGED

1 An attempt was made to change the key of a set or automatic subset that does not allow duplicates.

LIMITERROR

1 An attempt was made to store too many records in a structure.

2 An attempt was made to make too many counted links refer to one record. The count limit specified in DASDL was exceeded.

3 An attempt was made to put too many entries into a set or subset.

4 This error message is no longer in use.

5 An attempt was made to open too many partitions.

6 A partitioned structure is referenced in a DMUTILITY INITIALIZE statement.

7 Too many cursors are being used for this structure.

NORECORD

1 The current path of the record to be inserted was not valid.

2 The current path of the record to be removed was not valid.

3 The current path containing the link was not valid.

4 The current path of the record to be linked to was not valid.

5 The current path of the master record was not valid.

6 The current path of the record to be found was not valid.

NOTFOUND

1 No record satisfied the condition.

2 A NEXT statement was attempted when the cursor was at the last entry in the structure.

3 A PRIOR statement was attempted when the cursor was at the first entry in the structure.

4 The manual subset key did not correspond with the key in the data record.

5 No current record was present (DELETED).

6 The current path was undefined.

7 Either the master record pointer was undefined or the embedded structure had not been created.

8 The link was undefined.

9 The link did not agree with the data record.

10 The link did not exist in the current record. (The wrong type of variable-format record was specified.)

11 A select-text error occurred (cannot have record because select-text restriction failed).

12 An invalid index resulted from an OCCURS.

NOTLOCKED

1 A STORE statement was attempted but it was not preceded by a CREATE, LOCK/MODIFY, or STORE statement.

2 An attempt was made to modify detail when the master record was not locked.

OPENERROR

1 The Accessroutines were not present on disk.

2 An I/O error occurred on the Accessroutines code file.

3 This error message is no longer in use.

4 DMUTILITY tried to initialize when the data base was in use.

5 The run-time description of the data base did not match the compile-time description.

6 An Accessroutines error occurred on one or more structures.

7 The user program and the DMSUPPORT library were compiled with description files that had different time stamps.

8 This error message is no longer in use.

9 An attempt was made to open RECOVERY but RECOVERY was not necessary.

10 The program DSed on the open of the restart data set.

11 A read error occurred on segment 0 of the restart data set.

12 The program DSed when initializing the restart data set to empty.

13 The program DSed when changing the names of the restart data set.

14 The restart data set was not present.

15 An attempt was made to open RECOVERY but RECOVERY was already in progress.

16 A REQUIRED structure was not invoked.

17 An invoked structure was not found.

18 An attempt was made to open RECONSTRUCTION when RECONSTRUCTION was already in progress.

19 The RECONSTRUCTION was unsuccessful.

20 The DBS stack overflowed.

21 This error message is no longer in use.

22 This error message is no longer in use.

23 An attempt was made to initialize one or more partitions when the partitions already existed.

24 The program DSed or a file attribute error occurred when attempting to open a file.

25 An attempt was made to open a logical data base that did not exist.

26 A security error occurred when attempting to execute an OPEN statement.

27 The Accessroutines code file found is not the same as the code file in use by the running data base. The Accessroutines title should be specified in DASDL to uniquely locate the code file, or the incorrect copy should be removed from the opener's usercode and family. If the Accessroutines are copied while the data base is open, new openers will also get this error until the data base is closed.

28 The system was unable to read row 0 of the restart data set (must reconstruct).

29 The user program was compiled with a description file that had a higher update level than the description file used to compile the DMSUPPORT library.

30 This error message is no longer in use.

31 An I/O error occurred on the control file.

32 An unknown type of OPEN was attempted.

33 The data base was open when a HALT/LOAD occurred.

34 The audit trail could not be initialized.

35 There was insufficient memory for the DBS stack.

36 The Accessroutines code file was unacceptable.

37 An open error occurred on the control file.

38 The data base software did not match the format for the structure.

39 The control file was locked by an exclusive function.

40 The application program must be recompiled on a more recent release.

41 The user program and the DMSUPPORT library were compiled with different description files.

42 This error message is no longer in use.

44 The DBS was not visible from the program's processor.

45 This option was not permitted while TPS was active.

46 This option was not permitted until TPS RECOVERY was complete.

47 The initiation of the Accessroutines failed.

48 The application program DSed before the Accessroutines were frozen.

49 The control file was not present on disk (operator DSed NO FILE condition).

50 An I/O error occurred while reading block 0 of the control file.

51 This error message is no longer in use.

52 This error message is no longer in use.

53 This error message is no longer in use.

56 The Accessroutines code file did not support audit.

57 The Accessroutines code file did not support partitions.

58 The Accessroutines code file did not support local buffering.

59 The data base was already in use by an exclusive or single update user.

60 The data base was already in use when an open exclusive or single update was attempted.

61 An attempt was made to open a Semantic Information Manager (SIM) data base, but SIM has not been implemented on the system.

62 An attempt was made to update a SIM data base that can be accessed only for inquiry purposes.

63 An attempt was made to update a DMSII data base that can be accessed only for inquiry purposes.

64 An attempt was made to library-equate a new DMSUPPORT Library, but either there is an existing DMSUPPORT Library or the old DMSUPPORT Library is still being accessed.

65 The usercode under which the data base control file is stored is not a valid usercode for the system. Either make the usercode a valid system usercode, or move the control file to an existing valid system usercode.

66 The DMSUPPORT Library was not resident.

67 The DMSUPPORT Library needs to be recompiled.

68 The Audit Block Serial Number (ABSN) is approaching its maximum allowable value and must be reset. (For information on resetting the ABSN, refer to the DMSII Utilities Operations Guide.)

69 A rebuild or rollback procedure is in progress.

READONLY

1 A change was attempted while in INQUIRY mode.

2 An INSERT or REMOVE statement was attempted on a READONLY record.

3 A CREATE statement for a new partition was attempted while in INQUIRY mode.

SECURITYERROR

1 Verb security: the user was not permitted to use this function.

SYSTEMERROR

1 A FIND FIRST, NEXT, PRIOR, or LAST statement was attempted but not allowed on embedded STANDARD or COMPACT data sets.

2 A GENERATE statement was attempted but not allowed on this type of set.

3 The system attempted to read beyond the end-of-file.

4 An Accessroutines fault occurred during DMSFREE (ENDTRANSACTION or EOT).

5 An invalid calling sequence or other sequence of events.

6 Fatal software error.

A fatal software error is produced in the following situations:

1. Inconsistent usage of internal system locks.

2. Inconsistent linkage between blocks of data.

3. Information read from mass storage device is inconsistent with information requested.

7 Fatal integrity error.

A fatal integrity error is produced in the following situations:

1. An address of zero was encountered.

2. A block address was not on a block boundary.

3. A record address was not on a record boundary.

4. The value of a key in a set or subset did not match the value of the key item in the record to which the set pointed.

5. A set or an automatic subset pointed to a deleted record.

6. A record did not have an entry in a set or an automatic subset when it should have had such an entry.

8 Fatal I/O error.

A fatal I/O error occurs in the following situations:

1. The audit file could not be opened.

2. Recovery information could not be read or written.

3. An attempt was made to allocate or deallocate available space in a file.

9 Fatal resource error.

A fatal resource error is encountered in the following situations:

1. The main memory was exhausted.

2. There was not enough space in the restart data set.

10 Fatal unknown error.

This error occurs when ABORT recovery terminates on an error.

11 Fatal operator DSed data base error.

This error occurs if an attempt was made to open a data base during a time just after the operator had DSed the data base or the data base was being terminated due to some fatal error.

12 A software error occurred in the control file handler.

13 The partitions control in the control file were uninitialized.

14 The Accessroutines code file was compiled without this structure type.

VERSIONERROR

1 The link was unknown or deleted.

2 An attempt was made to reference a structure that had been deleted or changed since the program was compiled.

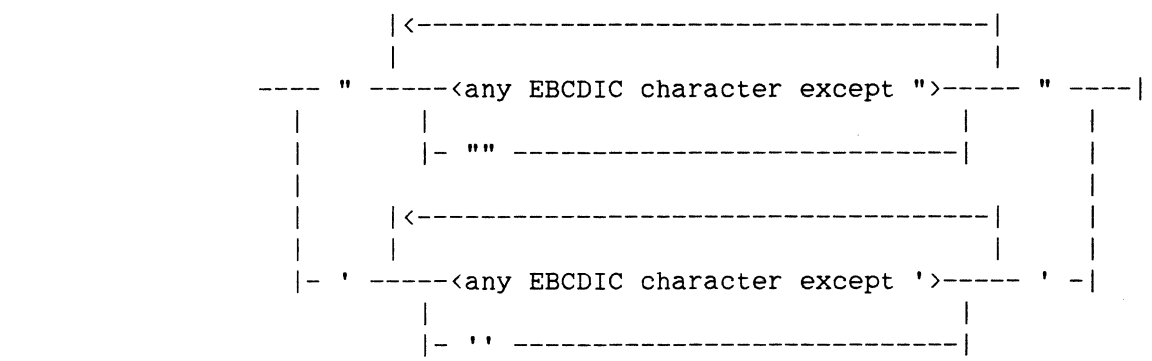
C SYNTAX DIAGRAMS FOR ENTRY POINT PARAMETERS

The following diagrams, listed in alphabetical order, describe the syntax of the parameters for DMINTERPRETER entry points. These diagrams repeat all of the diagrams given in the interface language sections of this manual (except for diagrams of the expression parameters), in addition to describing basic data base constructs that are used in the diagrams. The diagrams are the same for all interface languages (except for diagrams of the expression parameters).

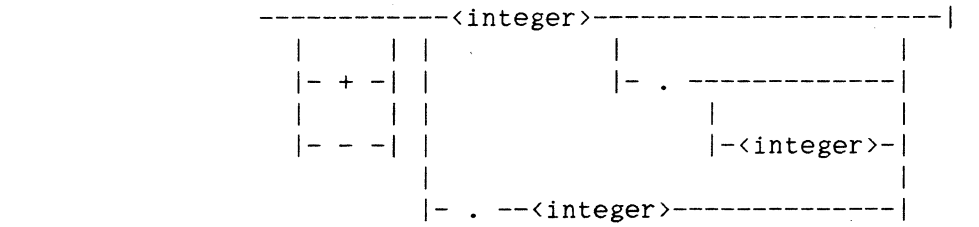
Syntax for the expression parameters varies from language to language, as explained in each interface language section of this manual. Refer to the indicated language manuals for further breakdowns of the basic language constructs that are used in the syntax diagrams for the expression parameters.

<alpha item name>

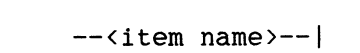
<alpha literal>



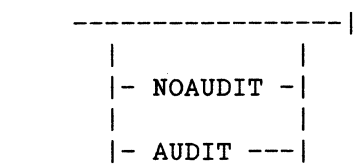
<arithmetic constant>



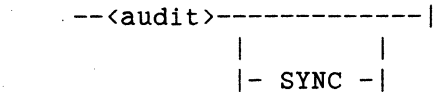
<arithmetic item name>



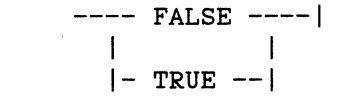
<audit>



<auditsync>

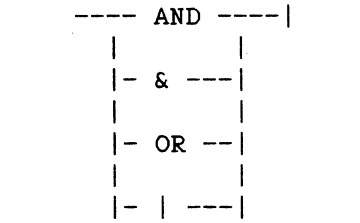


<Boolean constant>

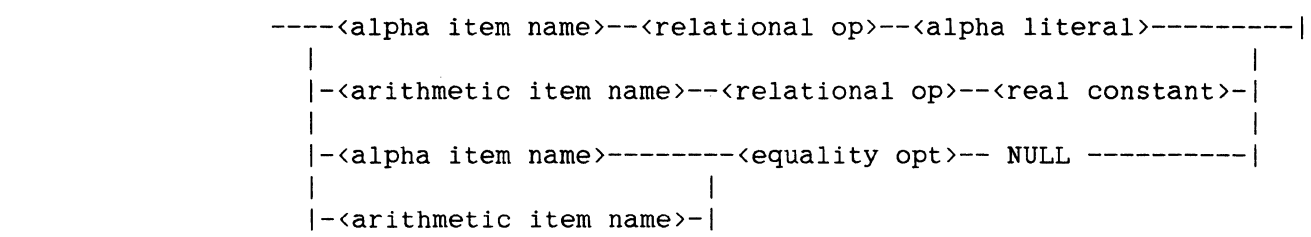


<Boolean item name>

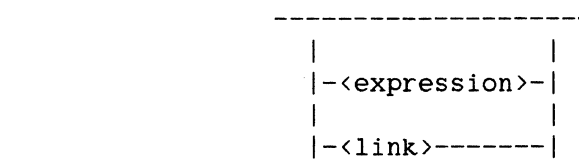
<Boolean op>



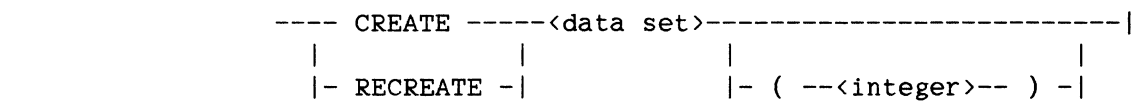
<Boolean primary>



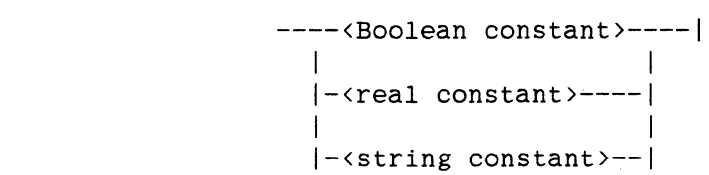
<condition>



<create statement>

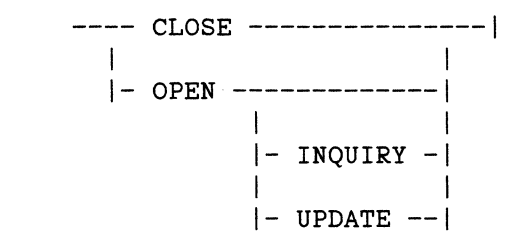


<data>

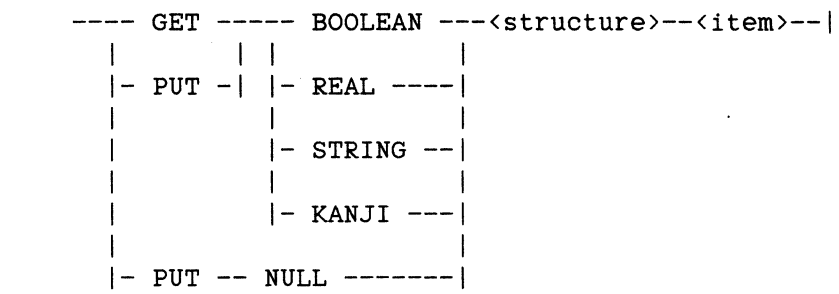


<data base>

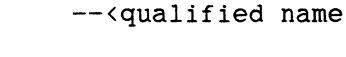
<data base statement>



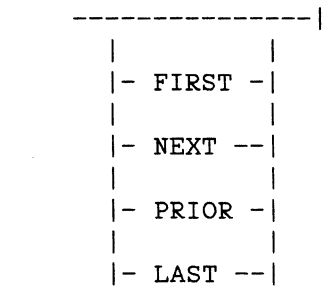
<data request>



<data set>

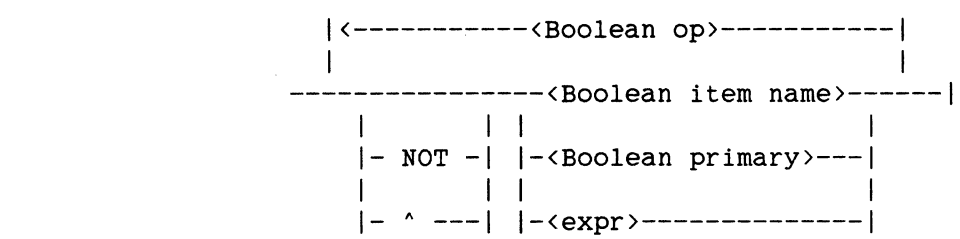


<direction>

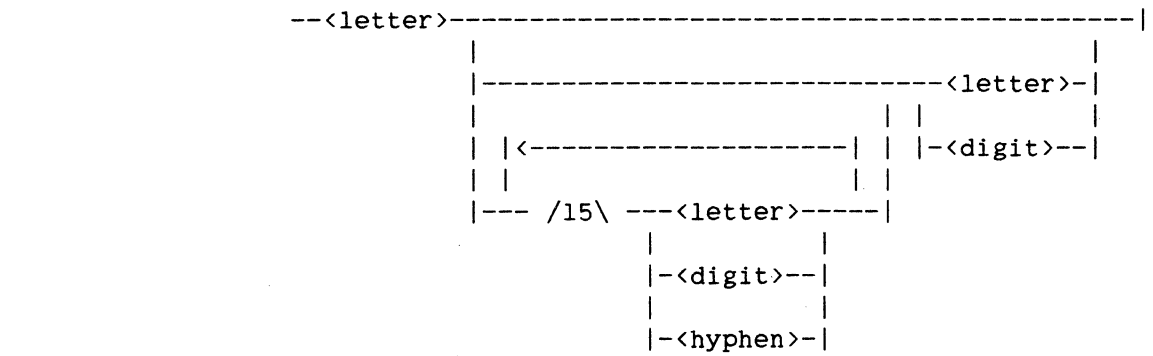


<equality op>

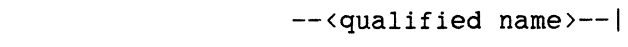
<expression>



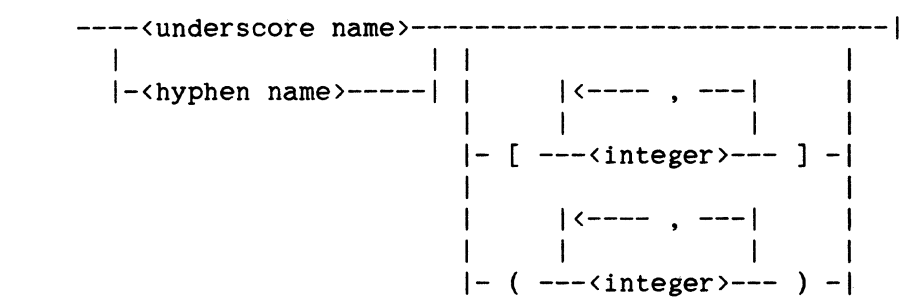
<hyphen name>



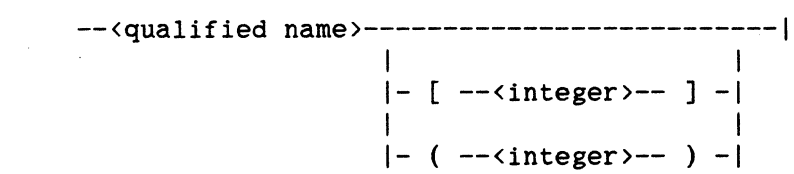
<index set>



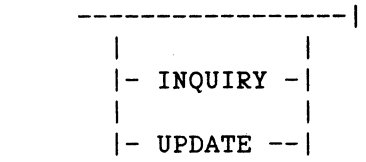
<item name>



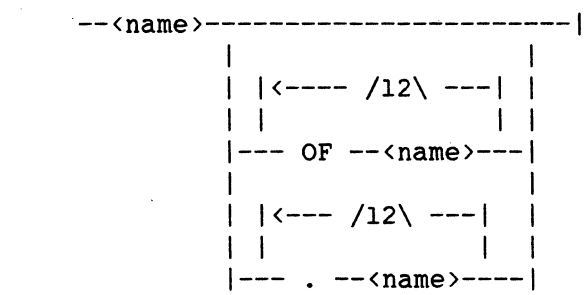
<link>



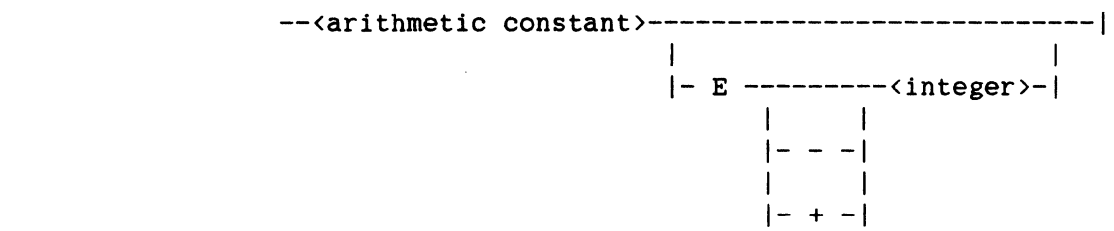
<open disposition>



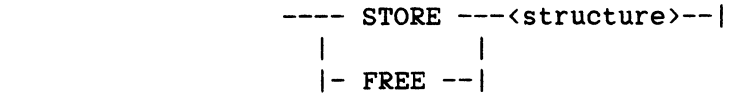
<qualified name>



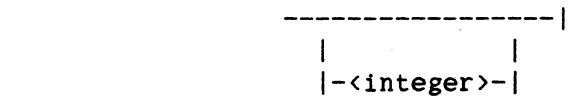
<real constant>



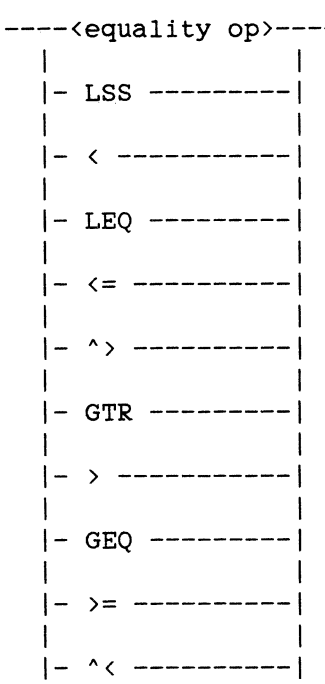
<record statement>



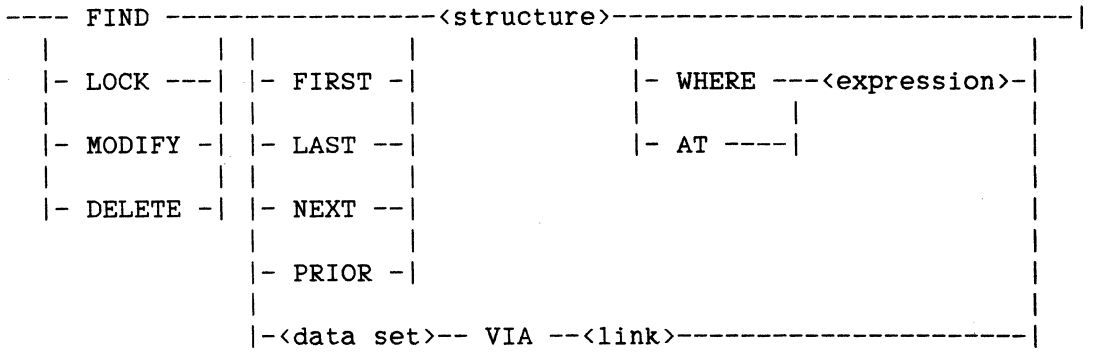
<record type>



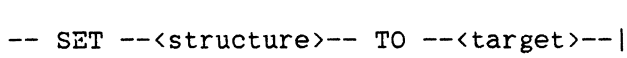
<relational op>



<search statement>

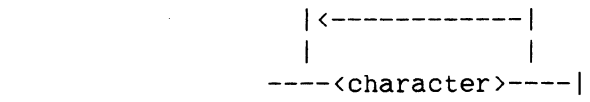


<set statement>

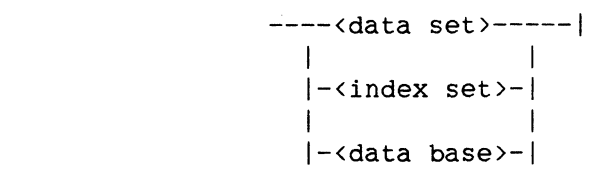


<status statement>

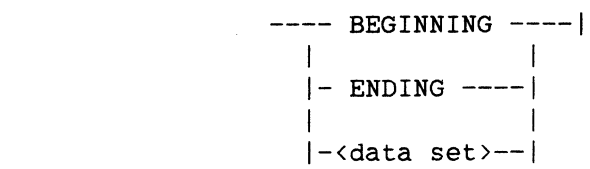
<string constant>



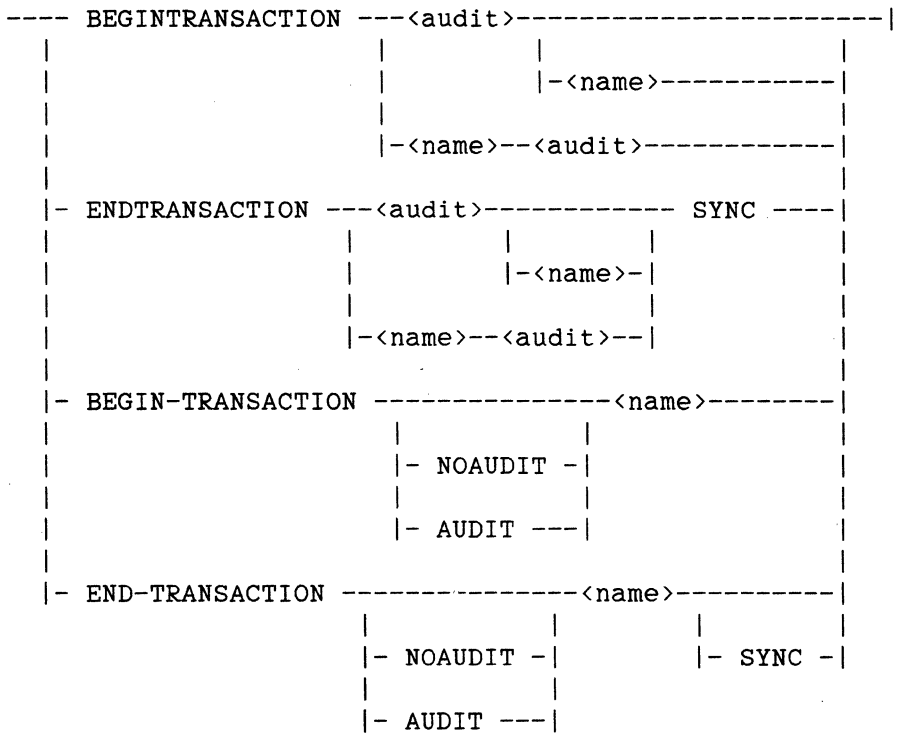
<structure>



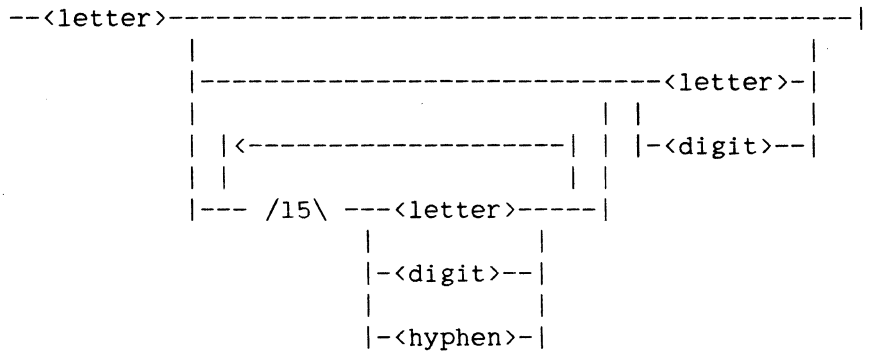
<target>



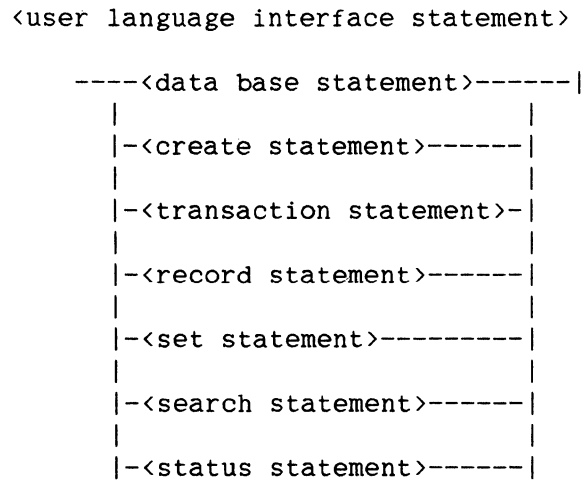
<transaction statement>



<underscore name>



Syntax Diagrams for Entry Point Parameters



D DBFIND ENHANCEMENTS

The DBFIND entry point of the DMINTERPRETER Library performs the same functions as the DMSII user language interface FIND statement. However, significant improvements over the FIND statement have been made to the DBFIND search algorithms. Some search optimization techniques previously available only to the DMSII INQUIRY program are now available for use by DMSII application programs via the interpretive interface.

NON-KEY ITEMS IN SELECTION EXPRESSIONS

The interpretive interface allows non-key items in conditions, whereas the user language interface allows only key and key data items in conditions. Relations involving non-key items might require a linear search, depending on other relations that are in the same condition. If an item is not REQUIRED, it may be tested for NULL. For NULL items, the relations "<item> NEQ <value>" and "<item> = NULL" are TRUE and all other relations are FALSE.

SELECTION EXPRESSION EVALUATION

Because the DMINTERPRETER Library is "interpretive," it must determine how to process selection expressions at run time. To reduce the overhead resulting from this run-time determination, the most recent DBFIND statement is saved in an internal form, and a selection expression must be reevaluated only when it changes. As a result, successive identical calls on the DBFIND entry point involve less overhead than they would ordinarily.

FINDING A DATA SET WITH SELECTION EXPRESSIONS

The interpretive interface does not require specification of a set or access for use in locating a record that meets a certain condition. Only the data set name is needed. (The DMSII user language interface does not allow selection expressions involving data sets.) When no set is specified and a selection expression is given, the interpretive interface uses the set or access that provides the most efficient search. Subsets are not considered. If no sets or accesses are available, a linear search is performed.

The benefits of this arrangement can be summarized as follows:

1. Less knowledge of the data base is required.

2. Data independence is improved.

3. If a new set is added that could be used more efficiently, old programs need not be rewritten to take advantage of this set.

As long as the search condition remains unchanged, the interpretive interface continues to use the selected set instead of the data set. The structure parameter of the DBFIND entry point can specify a data set even when the condition parameter is nonempty. By using a data set instead of an index set as a parameter for the DBFIND entry point, application programs can be protected from changes that add or delete sets from the data base.

When the search condition changes, the interpretive interface reevaluates the index set it chose to access the data set. If a different index set is chosen, a call on the DBFIND entry point could produce unexpected results. For example, a FIND NEXT operation might actually return a prior record. The interpretive interface returns an exception when such a situation is detected. This problem can be avoided by using any of the following methods:

1. Not changing the condition in the middle of a search on a data set.

2. Specifying an index set instead of a data set as a DBFIND parameter.

3. Calling DBSET or DBFIND with the direction parameter equal to FIRST or LAST before changing the condition parameter.

EARLY TERMINATION OF LINEAR SEARCHES

The interpretive interface performs linear searches in some cases, but it does not always search the entire structure being used. In many cases (for example, an LEQ on an ASCENDING key), the search can be terminated the first time the search condition fails. This form of linear search can, at times, be faster than a binary search.

MORE POWERFUL BINARY SEARCHES

To perform a binary search on a complex key, the DMSII user language interface requires an equality test for all but the most minor key specified. Otherwise, a linear search of the index is performed. But the interpretive interface can perform a binary search if all conditions that do not use EQL use the same binary-searchable inequality.

For example, given a set with PARTNO and QUANTITY as its first and second keys, the following DMSII user language interface statement produces a linear search:

```
FIND PART AT PARTNO > 10 AND QUANTITY > 20;
```

The following equivalent ALGOL statement for use with the interpretive interface is processed using a binary search:

```
STRING EMPTY;
EMPTY := " ";
DBFIND(EMPTY,"PART","PARTNO>10 AND QUANTITY>20");
```

The interpretive interface also performs binary searches on items inside a GROUP key. The following example illustrates the situation:

```
DASDL:  GROUPDATA DATA SET
@       (GROUPKEY GROUP(GROUPITEM NUMBER(2);
@                       OTHERITEM ALPHA(10)));
@       GROUPSET SET OF GROUPDATA KEY IS GROUPKEY;
```

The following DMSII user language interface statement produces a linear search of GROUPSET:

```
FIND NEXT GROUPSET AT GROUPITEM = 99;
```

The equivalent statement for the interpretive interface is processed using a binary search of GROUPSET:

```
DBFIND("NEXT","GROUPSET","GROUPITEM=99");
```

MIXED MODE SEARCHES

The interpretive interface can combine linear and binary searches to satisfy a condition. For example, a binary search might be used to find the beginning of a subsequent linear search. The linear search might then terminate early when the search condition becomes FALSE.

AUTOMATIC FIND KEY OF

A data set record is not actually moved into the DMSII user work area as a result of all calls on the DBFIND entry point. Instead, if a set is used, only the set entry is read and the data set read operation is simply marked as "pending." The interpretive interface completes the read if a statement is issued that requires one of the following:

1. Data that is not in the key area.

2. A correct path to the data.

As a result, a call on the DBFIND entry point that references only key or key data items behaves like a DMSII user language interface FIND KEY OF statement. "Get" operations of the interpretive interface (which move data from the the DMSII user work area to variables that have been declared by the programmer) that reference only key or key data items do not cause the data set record to be read. Consequently, the data set record might never need to be read.

E PROGRAMMING CONSIDERATIONS

When writing application programs that access the interpretive interface, following these recommendations can help reduce run-time overhead:

1. When programming in ALGOL or PL/I, pre-initialized string variables, instead of string expressions, should be used as parameters. This eliminates the need to allocate string temporaries in the calling procedure.

2. The DMINTERPRETER Library remembers the last structure on which it operated. Consequently, structure-switching overhead can be reduced if calls on one structure are not interspersed with consecutive calls on another structure.

3. Passing unnecessary blanks to the interpretive interface should be avoided. The use of blanks involves considerable extra overhead because additional procedures must be called, and string temporaries must be allocated during blank elimination. For example, DBSTORE("DS") is much faster than DBSTORE("@DS").

4. Because ALGOL and PL/I programs directly call the entry points that perform the functions of the interpretive interface, overhead can be minimized by programming in these languages rather than in others.

Because calls on DBBEGINTRANSACTION and DBENDTRANSACTION are ignored for unaudited databases, application programs can be simplified by unconditionally calling these entry points. The program can then be run against an audited or unaudited data base without alteration.

To properly compile a PL/I program that invokes either DBGETDOUBLE or DBPUTDOUBLE, an OPTIONS(DOUBLE) clause should be used in the outermost procedure declaration.

The DMINTERPRETER Library stack uses globals to retain information between calls. This greatly increases the speed of execution, but prevents multiprocessing. Thus, if the library declared by a parent task is visible to two or more offspring tasks, the offspring can receive unexpected results if they call the library simultaneously. Locking conventions within the DMINTERPRETER Library could prevent such confusion, but would slow down execution speed. Consequently, when the library is visible to multiple offspring tasks, the application program should implement its own locking protocols.

As with other tailored DMSII software, data base equation cannot be used when accessing the DMINTERPRETER Library. Instead, a separate DMINTERPRETER Library code file must be compiled for each data base.

F SAMPLE PROGRAMS

SAMPLE PROGRAM IN COBOL

The following is a sample of a COBOL application program.

```
************************
 IDENTIFICATION DIVISION.
************************
*PROGRAM-ID.
*AUTHOR.
*INSTALLATION.
*DATE-WRITTEN.
*DATE-COMPILED.
*SECURITY.     PUBLIC.
*REMARKS.
*THIS PROGRAM EXERCISES ENTRY POINTS OF THE DMINTERPRETER LIBRARY,
*WHICH HAS BEEN COMPILED FOR THE EMPJOB SAMPLE DATABASE.
**********************
 ENVIRONMENT DIVISION.
**********************
 CONFIGURATION SECTION.
*======================
 SOURCE-COMPUTER. B5900/B6000/B7000.
 OBJECT-COMPUTER. B5900/B6000/B7000.
*===================================
 INPUT-OUTPUT SECTION.
*====================
 FILE-CONTROL.
     SELECT REMOTEFILE ASSIGN TO REMOTE.
****************************************
 DATA DIVISION.
***************
 FILE SECTION.
 FD  REMOTEFILE
     RECORD CONTAINS 80 CHARACTERS
     VALUE OF TITLE IS "REMOTEFILE".
 01  REMOTE-REC           PIC X(80).
*====================================
 WORKING-STORAGE SECTION.
*DMINTERPRETER CODE FILE.
*-----------------------
 01  DMI-NAME             PIC X(24) VALUE "DMINTERPRETER/EMPJOB".
*DMI VOCABULARY.
*---------------
 01  DATA-SET-NAME        PIC X(17) VALUE IS "EMP".
 01  ITEM-NAME            PIC X(17) VALUE IS SPACES.
 01  FIRST-1              PIC X(5) VALUE "FIRST".
 01  PRIOR-1              PIC X(5) VALUE "PRIOR".
 01  NEXT-1               PIC X(4) VALUE "NEXT".
```

```
 01  P-AUDIT              PIC X(5) VALUE "AUDIT".
 01  N-AUDIT              PIC X(8) VALUE "NOAUDIT".
 01  OPEN-TYPE            PIC X(6) VALUE "UPDATE".
 01  BEGIN-1              PIC X(9) VALUE "BEGINNING".
 01  CONDITION-1          PIC X(9) VALUE "EMP-NO=11".
 01  SPACE-1              PIC X(1) VALUE " ".
 01  DATA-BUFFER          PIC X(10) VALUE IS SPACES.
 01  DATA-REQUEST         PIC X(40) VALUE IS SPACES.
 01  VAR-1                PIC X(10) VALUE IS SPACES.
 77  VAR-2                COMP-4.
 77  VAR-3                COMP-5.

*
*  EXCEPTION AREA
*
 01  RESULT                PIC 9(1) COMP-2 VALUE IS 0.
 01  EXCEPTION-NAME        PIC X(14).
 01  EXCEPTION-MSG         PIC X(80).
 01  EXCEPTION-WORD        PIC 9(12) COMP-2.

******************** EXCEPTION-WORD FIELDS***********************
*
*    02  STRUCTURE         PIC 9(4) COMP-2.
*    02  CATEGORY          PIC 9(3) COMP-2.
*    02  SUB-CATEGORY      PIC 9(4) COMP-2.
*    02  FLAG              PIC 9(1) COMP-2.
*
*****************************************************************
 PROCEDURE DIVISION.
 MAIN-SECTION.

*****************************************************************
*                       BODY OF PROGRAM                         *
*****************************************************************
*
****************** LINKING TO DMI LIBRARY ***********************
*
     CHANGE ATTRIBUTE TITLE OF "DMINTERPRETER"
         TO DMI-NAME.

*****************************************************************
     OPEN OUTPUT REMOTEFILE.

*****************************************************************
*                    OPENING THE DATABASE                       *
*****************************************************************

     CALL "DBOPEN OF DMINTERPRETER"
         USING OPEN-TYPE
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.
```

```
*****************************************************************
*                      DELETING A RECORD                        *
*****************************************************************

     CALL "DBBEGINTRANSACTION OF DMINTERPRETER"
         USING P-AUDIT
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.
     CALL "DBDELETE  OF DMINTERPRETER"
         USING FIRST-1, DATA-SET-NAME, CONDITION-1
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.
     MOVE "EMP-NO=12" TO CONDITION-1.
     CALL "DBDELETE  OF DMINTERPRETER"
         USING FIRST-1, DATA-SET-NAME, CONDITION-1
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.
     CALL "DBENDTRANSACTION OF DMINTERPRETER"
         USING P-AUDIT
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

*****************************************************************
*                      CREATING A RECORD                        *
*****************************************************************

     CALL "DBCREATE OF DMINTERPRETER"
         USING DATA-SET-NAME, SPACE-1
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

*****************************************************************
*                     FILLING IN A RECORD                       *
*****************************************************************

*
*            USING  DBPUTREAL ENTRY POINT
*
     MOVE "EMP-NO" TO ITEM-NAME.
     MOVE  11  TO VAR-2.
     CALL "DBPUTREAL OF DMINTERPRETER"
         USING DATA-SET-NAME,ITEM-NAME,VAR-2
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

     MOVE "EMP-SALARY" TO ITEM-NAME.
     MOVE 9001.90 TO VAR-2.
```

```
      CALL "DBPUTREAL OF DMINTERPRETER"
          USING DATA-SET-NAME,ITEM-NAME,VAR-2
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
*
*             USING DBPUTDOUBLE ENTRY POINT
*
      MOVE "EMP-SSN" TO ITEM-NAME.
      MOVE  573230911 TO VAR-3.
      CALL "DBPUTDOUBLE  OF DMINTERPRETER"
          USING DATA-SET-NAME,ITEM-NAME,VAR-3
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
*
*              USING DBDATA ENTRY POINT
*
      MOVE "PUT REAL EMP EMP-DATE-HIRED"TO DATA-REQUEST.
      MOVE "811223" TO DATA-BUFFER
      CALL "DBDATA OF DMINTERPRETER"
          USING DATA-REQUEST,DATA-BUFFER
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
      MOVE "PUT STRING EMP EMP-LNAME" TO DATA-REQUEST.
      MOVE "STACK" TO DATA-BUFFER.
      CALL "DBDATA OF DMINTERPRETER"
          USING DATA-REQUEST,DATA-BUFFER
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
*
*              USING DBPUTDISPLAY ENTRY POINT
*
      MOVE "EMP-FNAME" TO ITEM-NAME.
      MOVE "CATHY" TO VAR-1.
      CALL "DBPUTDISPLAY OF DMINTERPRETER"
          USING DATA-SET-NAME,ITEM-NAME,VAR-1
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
*
*              USING DBPUTBOOLEAN
*
      MOVE "EMP-SEC-CLEAR" TO ITEM-NAME.
      MOVE "TRUE" TO VAR-1.
      CALL "DBPUTBOOLEAN  OF DMINTERPRETER"
          USING DATA-SET-NAME,ITEM-NAME,VAR-1
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
```

```
*****************************************************************
*                RECORD SET-UP IN USER WORK AREA                *
*****************************************************************
* EMP     EMP      EMP      EMP          EMP          EMP       EMP    *
*  NO    LNAME    FNAME     SSN       DATE-HIRED   SEC-CLEAR   SALARY *
*---------------------------------------------------------------------*
*  11    STACK    CATHY   573230911    811223       TRUE      9001.90 *
*****************************************************************


*****************************************************************
*                        STORING A RECORD                       *
*****************************************************************

     CALL "DBBEGINTRANSACTION OF DMINTERPRETER"
         USING P-AUDIT
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.
     CALL "DBSTORE OF DMINTERPRETER"
         USING DATA-SET-NAME
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.
     CALL "DBENDTRANSACTION OF DMINTERPRETER"
         USING P-AUDIT
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

*****************************************************************
*                      RECREATING A RECORD                      *
*****************************************************************

     CALL "DBRECREATE OF DMINTERPRETER"
         USING DATA-SET-NAME, SPACE-1
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

     MOVE "EMP-NO" TO ITEM-NAME.
     MOVE  12  TO VAR-2.
     CALL "DBPUTREAL OF DMINTERPRETER"
         USING DATA-SET-NAME,ITEM-NAME,VAR-2
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

     MOVE "EMP-SSN" TO ITEM-NAME.
     MOVE  549927328 TO VAR-3.
     CALL "DBPUTDOUBLE  OF DMINTERPRETER"
         USING DATA-SET-NAME,ITEM-NAME,VAR-3
         GIVING RESULT.
```

```
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.

      MOVE "PUT REAL EMP EMP-DATE-HIRED"TO DATA-REQUEST.
      MOVE "811228" TO DATA-BUFFER
      CALL "DBDATA OF DMINTERPRETER"
          USING DATA-REQUEST,DATA-BUFFER
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
      MOVE "PUT STRING EMP EMP-LNAME" TO DATA-REQUEST.
      MOVE "WOLVERTON" TO DATA-BUFFER.
      CALL "DBDATA OF DMINTERPRETER"
          USING DATA-REQUEST,DATA-BUFFER
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.

      MOVE "EMP-FNAME" TO ITEM-NAME.
      MOVE "LAURA" TO VAR-1.
      CALL "DBPUTDISPLAY OF DMINTERPRETER"
          USING DATA-SET-NAME,ITEM-NAME,VAR-1
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.

*****************************************************************
*          SET-UP OF RE-CREATED RECORD IN USER WORK AREA        *
*****************************************************************
* EMP    EMP       EMP       EMP        EMP          EMP        EMP   *
*  NO   LNAME     FNAME     SSN     DATE-HIRED   SEC-CLEAR   SALARY  *
*---------------------------------------------------------------------*
* 12  WOLVERTON  LAURA  549927328   811228       TRUE      9001.90 *
*****************************************************************

      CALL "DBBEGINTRANSACTION OF DMINTERPRETER"
          USING P-AUDIT
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
      CALL "DBSTORE OF DMINTERPRETER"
          USING DATA-SET-NAME
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
      CALL "DBENDTRANSACTION OF DMINTERPRETER"
          USING P-AUDIT
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
```

```
*****************************************************************
*              SETTING DATA SET TO THE BEGINNING                *
*****************************************************************

     CALL "DBSET OF DMINTERPRETER"
         USING DATA-SET-NAME, BEGIN-1
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

*****************************************************************
*                         FINDING A RECORD                      *
*****************************************************************

     CALL "DBFIND OF DMINTERPRETER"
         USING FIRST-1, DATA-SET-NAME, CONDITION-1
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

*****************************************************************
*                         LOCKING A RECORD                      *
*****************************************************************

     CALL "DBLOCK OF DMINTERPRETER"
         USING FIRST-1, DATA-SET-NAME, CONDITION-1
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

*****************************************************************
*                    VERIFYING ITEMS IN RECORD                  *
*****************************************************************
*
*               USING DBGETREAL ENTRY POINT
*
     MOVE "EMP-NO" TO ITEM-NAME.
     CALL "DBGETREAL OF DMINTERPRETER"
         USING DATA-SET-NAME,ITEM-NAME,VAR-2
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

     MOVE "EMP-SALARY" TO ITEM-NAME.
     CALL "DBGETREAL OF DMINTERPRETER"
         USING DATA-SET-NAME,ITEM-NAME,VAR-2
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.
*
*         USING DBGETDOUBLE ENTRY POINT
*
     MOVE "EMP-SSN" TO ITEM-NAME.
```

```
      CALL "DBGETDOUBLE  OF DMINTERPRETER"
          USING DATA-SET-NAME,ITEM-NAME,VAR-3
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
*
*             USING DBDATA ENTRY POINT
*
      MOVE "GET REAL EMP EMP-DATE-HIRED" TO DATA-REQUEST.
      CALL "DBDATA OF DMINTERPRETER"
          USING DATA-REQUEST,DATA-BUFFER
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
      MOVE "GET STRING EMP EMP-LNAME" TO DATA-REQUEST.
      CALL "DBDATA OF DMINTERPRETER"
          USING DATA-REQUEST,DATA-BUFFER
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
*
*         USING DBGETDISPLAY ENTRY POINT
*
      MOVE "EMP-FNAME" TO ITEM-NAME.
      CALL "DBGETDISPLAY OF DMINTERPRETER"
          USING DATA-SET-NAME,ITEM-NAME,VAR-1
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
*
*         USING DBGETBOOLEAN
*
      MOVE "EMP-SEC-CLEAR" TO ITEM-NAME.
      CALL "DBGETBOOLEAN OF DMINTERPRETER"
          USING DATA-SET-NAME,ITEM-NAME,VAR-1
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.

*****************************************************************
*                      FREEING THE RECORD                       *
*****************************************************************

      CALL "DBFREE OF DMINTERPRETER"
          USING DATA-SET-NAME
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
```

```
*****************************************************************
*                      CAUSING AN EXCEPTION                     *
*****************************************************************

     CALL "DBFIND OF DMINTERPRETER"
         USING FIRST-1, DATA-SET-NAME, SPACE-1
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.
     CALL "DBFIND OF DMINTERPRETER"
         USING PRIOR-1, DATA-SET-NAME, SPACE-1
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

*****************************************************************
*                          USING DBVERB                         *
*****************************************************************
*
*        USING DBVERB ON A SET STATEMENT
*
      MOVE "SET EMP TO BEGINNING" TO DATA-REQUEST.
      CALL "DBVERB OF DMINTERPRETER"
          USING DATA-REQUEST
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
*
*         USING DBVERB ON A SEARCH STATEMENT
*
      MOVE "FIND FIRST EMP AT EMP-NO=11" TO DATA-REQUEST.
      CALL "DBVERB OF DMINTERPRETER"
           USING DATA-REQUEST
           GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.

*****************************************************************
 *                        CLOSING THE DATA BASE                  *
*****************************************************************

     CALL "DBCLOSE OF DMINTERPRETER"
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

*****************************************************************

     CLOSE REMOTEFILE.
     STOP RUN.

*****************************************************************
```

```
 EXCEPTION-HANDLER.

     CALL "DBSTATUS OF DMINTERPRETER"
         USING EXCEPTION-WORD.
     CALL "DBEXCEPTIONNAME OF DMINTERPRETER"
         USING EXCEPTION-WORD, EXCEPTION-NAME.
     MOVE EXCEPTION-NAME TO REMOTE-REC.
     WRITE REMOTE-REC.
     CALL "DBEXCEPTIONTEXT OF DMINTERPRETER"
         USING EXCEPTION-WORD, EXCEPTION-MSG.
     MOVE EXCEPTION-MSG TO REMOTE-REC.
     WRITE REMOTE-REC.

*****************************************************************

 END-OF-PROGRAM.
     EXIT PROGRAM.

*****************************************************************
```

SAMPLE COBOL74 PROGRAM

The following is a sample COBOL74 application program.

```
 ************************
  IDENTIFICATION DIVISION.
 ************************
 *PROGRAM-ID.
 *AUTHOR.
 *INSTALLATION.
 *DATE-WRITTEN.
 *DATE-COMPILED.
 *SECURITY.     PUBLIC.
 *REMARKS.
 *THIS PROGRAM EXERCISES ENTRY POINTS OF THE DMINTERPRETER LIBRARY,
 *WHICH HAS BEEN COMPILED FOR THE EMPJOB SAMPLE DATABASE.
 **********************
  ENVIRONMENT DIVISION.
 **********************
  CONFIGURATION SECTION.
 *======================
  SOURCE-COMPUTER. B5900/B6000/B7000.
  OBJECT-COMPUTER. B5900/B6000/B7000.
 *====================================
  INPUT-OUTPUT SECTION.
 *====================
  FILE-CONTROL.
      SELECT REMOTEFILE ASSIGN TO REMOTE.
 ****************************************
  DATA DIVISION.
 ***************
  FILE SECTION.
  FD  REMOTEFILE
      RECORD CONTAINS 80 CHARACTERS
      VALUE OF TITLE IS "REMOTEFILE".
  01  REMOTE-REC           PIC X(80).
 *====================================
  WORKING-STORAGE SECTION.
 *DMINTERPRETER CODE FILE.
 *-----------------------
  01  DMI-NAME             PIC X(24) VALUE "DMINTERPRETER/EMPJOB".
 *DMI VOCABULARY.
 *---------------
  01  DATA-SET-NAME        PIC X(17) VALUE IS "EMP".
  01  ITEM-NAME            PIC X(17) VALUE IS SPACES.
  01  FIRST-1              PIC X(5) VALUE "FIRST".
  01  PRIOR-1              PIC X(5) VALUE "PRIOR".
  01  NEXT-1               PIC X(4) VALUE "NEXT".
  01  P-AUDIT              PIC X(5) VALUE "AUDIT".
```

```
 01  N-AUDIT              PIC X(8) VALUE "NOAUDIT".
 01  OPEN-TYPE            PIC X(6) VALUE "UPDATE".
 01  BEGIN-1              PIC X(9) VALUE "BEGINNING".
 01  CONDITION-1          PIC X(9) VALUE "EMP-NO=11".
 01  SPACE-1              PIC X(1) VALUE " ".
 01  DATA-BUFFER          PIC X(10) VALUE IS SPACES.
 01  DATA-REQUEST         PIC X(40) VALUE IS SPACES.
 01  VAR-1                PIC X(10) VALUE IS SPACES.

*
*  EXCEPTION AREA
*
 01  RESULT                PIC 9(1) COMP VALUE IS 0.
 01  EXCEPTION-NAME        PIC X(14).
 01  EXCEPTION-MSG         PIC X(80).
 01  EXCEPTION-WORD        PIC 9(12) COMP.

******************** EXCEPTION-WORD FIELDS***********************

*    02  STRUCTURE         PIC 9(4) COMP.
*    02  CATEGORY          PIC 9(3) COMP.
*    02  SUB-CATEGORY      PIC 9(4) COMP.
*    02  FLAG              PIC 9(1) COMP.
*
*****************************************************************

 PROCEDURE DIVISION.
 MAIN-SECTION.

*****************************************************************
*                       BODY OF PROGRAM                         *
*****************************************************************

****************** LINKING TO DMI LIBRARY ***********************

     CHANGE ATTRIBUTE TITLE OF "DMINTERPRETER"
         TO DMI-NAME.
*****************************************************************

     OPEN OUTPUT REMOTEFILE.

*****************************************************************
*                     OPENING THE DATABASE                      *
*****************************************************************

     CALL "DBOPEN OF DMINTERPRETER"
         USING OPEN-TYPE
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.
```

```
*****************************************************************
                        DELETING A RECORD                        *
*****************************************************************

     CALL "DBBEGINTRANSACTION OF DMINTERPRETER"
         USING P-AUDIT
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.
     CALL "DBDELETE  OF DMINTERPRETER"
         USING FIRST-1, DATA-SET-NAME, CONDITION-1
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.
     MOVE "EMP-NO=12" TO CONDITION-1.
     CALL "DBDELETE  OF DMINTERPRETER"
         USING FIRST-1, DATA-SET-NAME, CONDITION-1
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.
     CALL "DBENDTRANSACTION OF DMINTERPRETER"
         USING P-AUDIT
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

*****************************************************************
*                      CREATING A RECORD                        *
*****************************************************************

     CALL "DBCREATE OF DMINTERPRETER"
         USING DATA-SET-NAME, SPACE-1
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

*****************************************************************
*                      FILLING  IN A RECORD                     *
*****************************************************************
*
*
*             USING DBDATA ENTRY POINT
*
     MOVE "PUT REAL EMP EMP-NO" TO DATA-REQUEST.
     MOVE "11" TO DATA-BUFFER.
     CALL "DBDATA OF DMINTERPRETER"
         USING DATA-REQUEST,DATA-BUFFER
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

     MOVE "PUT REAL EMP EMP-SALARY" TO DATA-REQUEST.
     MOVE "9001.90" TO DATA-BUFFER.
```

```
      CALL "DBDATA OF DMINTERPRETER"
          USING DATA-REQUEST,DATA-BUFFER
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.

      MOVE "PUT REAL EMP EMP-SSN" TO DATA-REQUEST.
      MOVE "573230911" TO DATA-BUFFER.
      CALL "DBDATA OF DMINTERPRETER"
          USING DATA-REQUEST,DATA-BUFFER
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.

      MOVE "PUT REAL EMP EMP-DATE-HIRED"TO DATA-REQUEST.
      MOVE "811223" TO DATA-BUFFER
      CALL "DBDATA OF DMINTERPRETER"
          USING DATA-REQUEST,DATA-BUFFER
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.

      MOVE "PUT STRING EMP EMP-LNAME" TO DATA-REQUEST.
      MOVE "STACK" TO DATA-BUFFER.
      CALL "DBDATA OF DMINTERPRETER"
          USING DATA-REQUEST,DATA-BUFFER
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
*
*             USING DBPUTDISPLAY ENTRY POINT
*
      MOVE "EMP-FNAME" TO ITEM-NAME.
      MOVE "CATHY" TO VAR-1.
      CALL "DBPUTDISPLAY OF DMINTERPRETER"
          USING DATA-SET-NAME,ITEM-NAME,VAR-1
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
*
*             USING DBPUTBOOLEAN
*
      MOVE "EMP-SEC-CLEAR" TO ITEM-NAME.
      MOVE "TRUE" TO VAR-1.
      CALL "DBPUTBOOLEAN  OF DMINTERPRETER"
          USING DATA-SET-NAME,ITEM-NAME,VAR-1
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
```

```
*****************************************************************
*                RECORD SET-UP IN USER WORK AREA                *
*****************************************************************
* EMP    EMP      EMP       EMP         EMP          EMP       EMP    *
*  NO   LNAME    FNAME      SSN      DATE-HIRED   SEC-CLEAR   SALARY  *
*---------------------------------------------------------------------*
*  11   STACK    CATHY   573230911    811223        TRUE     9001.90 *
*****************************************************************


*****************************************************************
*                        STORING A RECORD                       *
*****************************************************************

     CALL "DBBEGINTRANSACTION OF DMINTERPRETER"
         USING P-AUDIT
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.
     CALL "DBSTORE OF DMINTERPRETER"
         USING DATA-SET-NAME
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.
     CALL "DBENDTRANSACTION OF DMINTERPRETER"
         USING P-AUDIT
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

*****************************************************************
*                      RECREATING A RECORD                      *
*****************************************************************

     CALL "DBRECREATE OF DMINTERPRETER"
         USING DATA-SET-NAME, SPACE-1
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

     MOVE "PUT REAL EMP EMP-NO" TO DATA-REQUEST.
     MOVE "12" TO DATA-BUFFER.
     CALL "DBDATA OF DMINTERPRETER"
         USING DATA-REQUEST,DATA-BUFFER
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

     MOVE "PUT REAL EMP EMP-SSN" TO DATA-REQUEST.
     MOVE "549927328" TO DATA-BUFFER.
     CALL "DBDATA OF DMINTERPRETER"
         USING DATA-REQUEST,DATA-BUFFER
         GIVING RESULT.
```

```
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.

      MOVE "PUT REAL EMP EMP-DATE-HIRED"TO DATA-REQUEST.
      MOVE "811228" TO DATA-BUFFER
      CALL "DBDATA OF DMINTERPRETER"
          USING DATA-REQUEST,DATA-BUFFER
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.

      MOVE "PUT STRING EMP EMP-LNAME" TO DATA-REQUEST.
      MOVE "WOLVERTON" TO DATA-BUFFER.
      CALL "DBDATA OF DMINTERPRETER"
          USING DATA-REQUEST,DATA-BUFFER
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.

      MOVE "EMP-FNAME" TO ITEM-NAME.
      MOVE "LAURA" TO VAR-1.
      CALL "DBPUTDISPLAY OF DMINTERPRETER"
          USING DATA-SET-NAME,ITEM-NAME,VAR-1
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.

*****************************************************************
*        SET-UP OF RE-CREATED RECORD IN USER WORK AREA          *
*****************************************************************
* EMP    EMP       EMP       EMP        EMP          EMP       EMP   *
*  NO   LNAME     FNAME      SSN     DATE-HIRED   SEC-CLEAR   SALARY *
*---------------------------------------------------------------*
* 12  WOLVERTON  LAURA  549927328   811228       TRUE      9001.90 *
*****************************************************************

      CALL "DBBEGINTRANSACTION OF DMINTERPRETER"
          USING P-AUDIT
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
      CALL "DBSTORE OF DMINTERPRETER"
          USING DATA-SET-NAME
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
      CALL "DBENDTRANSACTION OF DMINTERPRETER"
          USING P-AUDIT
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
```

```
*****************************************************************
*                SETTING DATA SET TO THE BEGINNING              *
*****************************************************************

     CALL "DBSET OF DMINTERPRETER"
         USING DATA-SET-NAME, BEGIN-1
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

*****************************************************************
*                        FINDING  A RECORD                      *
*****************************************************************

     CALL "DBFIND OF DMINTERPRETER"
         USING FIRST-1, DATA-SET-NAME, CONDITION-1
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

*****************************************************************
*                       LOCKING A RECORD                        *
*****************************************************************

     CALL "DBLOCK OF DMINTERPRETER"
         USING FIRST-1, DATA-SET-NAME, CONDITION-1
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

*****************************************************************
*                    VERIFYING ITEMS IN RECORD                  *
*****************************************************************
*
*             USING DBDATA ENTRY POINT
*
     MOVE "GET REAL EMP EMP-NO" TO DATA-REQUEST.
     CALL "DBDATA OF DMINTERPRETER"
         USING DATA-REQUEST,DATA-BUFFER
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

     MOVE "GET REAL EMP EMP-SALARY" TO DATA-REQUEST.
     CALL "DBDATA OF DMINTERPRETER"
         USING DATA-REQUEST,DATA-BUFFER
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

     MOVE "GET REAL EMP EMP-SSN" TO DATA-REQUEST.
     CALL "DBDATA OF DMINTERPRETER"
         USING DATA-REQUEST,DATA-BUFFER
```

```
        GIVING RESULT.
    IF RESULT = 1
        PERFORM EXCEPTION-HANDLER.

    MOVE "GET REAL EMP EMP-DATE-HIRED" TO DATA-REQUEST.
    CALL "DBDATA OF DMINTERPRETER"
        USING DATA-REQUEST,DATA-BUFFER
        GIVING RESULT.
    IF RESULT = 1
        PERFORM EXCEPTION-HANDLER.

    MOVE "GET STRING EMP EMP-LNAME" TO DATA-REQUEST.
    CALL "DBDATA OF DMINTERPRETER"
        USING DATA-REQUEST,DATA-BUFFER
        GIVING RESULT.
    IF RESULT = 1
        PERFORM EXCEPTION-HANDLER.
*
*       USING DBGETDISPLAY ENTRY POINT
*
    MOVE "EMP-FNAME" TO ITEM-NAME.
    CALL "DBGETDISPLAY OF DMINTERPRETER"
        USING DATA-SET-NAME,ITEM-NAME,VAR-1
        GIVING RESULT.
    IF RESULT = 1
        PERFORM EXCEPTION-HANDLER.
*
*       USING DBGETBOOLEAN
*
    MOVE "EMP-SEC-CLEAR" TO ITEM-NAME.
    CALL "DBGETBOOLEAN OF DMINTERPRETER"
        USING DATA-SET-NAME,ITEM-NAME,VAR-1
        GIVING RESULT.
    IF RESULT = 1
        PERFORM EXCEPTION-HANDLER.

*****************************************************************
*                    FREEING THE RECORD                         *
*****************************************************************

    CALL "DBFREE OF DMINTERPRETER"
        USING DATA-SET-NAME
        GIVING RESULT.
    IF RESULT = 1
        PERFORM EXCEPTION-HANDLER.

*****************************************************************
*                   CAUSING AN EXCEPTION                        *
*****************************************************************

    CALL "DBFIND OF DMINTERPRETER"
        USING FIRST-1, DATA-SET-NAME, SPACE-1
        GIVING RESULT.
```

```
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.
     CALL "DBFIND OF DMINTERPRETER"
         USING PRIOR-1, DATA-SET-NAME, SPACE-1
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

*****************************************************************
*                          USING DBVERB                         *
*****************************************************************
*
*        USING DBVERB ON A SET STATEMENT
*
      MOVE "SET EMP TO BEGINNING" TO DATA-REQUEST.
      CALL "DBVERB OF DMINTERPRETER"
          USING DATA-REQUEST
          GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.
*
*         USING DBVERB ON A SEARCH STATEMENT
*
      MOVE "FIND FIRST EMP AT EMP-NO=11" TO DATA-REQUEST.
      CALL "DBVERB OF DMINTERPRETER"
           USING DATA-REQUEST
           GIVING RESULT.
      IF RESULT = 1
          PERFORM EXCEPTION-HANDLER.

*****************************************************************
 *                        CLOSING THE DATA BASE                  *
*****************************************************************

     CALL "DBCLOSE OF DMINTERPRETER"
         GIVING RESULT.
     IF RESULT = 1
         PERFORM EXCEPTION-HANDLER.

*****************************************************************

     CLOSE REMOTEFILE.
     STOP RUN.

*****************************************************************

 EXCEPTION-HANDLER.

     CALL "DBSTATUS OF DMINTERPRETER"
         USING EXCEPTION-WORD.
     CALL "DBEXCEPTIONNAME OF DMINTERPRETER"
         USING EXCEPTION-WORD, EXCEPTION-NAME.
     MOVE EXCEPTION-NAME TO REMOTE-REC.
```

```
      WRITE REMOTE-REC.
      CALL "DBEXCEPTIONTEXT OF DMINTERPRETER"
          USING EXCEPTION-WORD, EXCEPTION-MSG.
      MOVE EXCEPTION-MSG TO REMOTE-REC.
      WRITE REMOTE-REC.

 ****************************************************************

  END-OF-PROGRAM.
      EXIT PROGRAM.

 ****************************************************************
```

SAMPLE ALGOL PROGRAM

This is a sample of an ALGOL program.

```
BEGIN
FILE REMOTEFILE
               (KIND = REMOTE
               ,UNITS=CHARACTERS
               ,MAXRECSIZE=1920
               ,MINRECSIZE=1
               ,FILETYPE=3
               );
%$ INCLUDE DMISYM="DATABASE/DMINTERPRETER" 20100000-20199999

%*********************************************************************%
%* In this program, the ALGOL declaration of library entry points has *%
%* been included from the DATABASE/DMINTERPRETER symbolic file,       *%
%* rather than explicitly declaring each entry point used.            *%
%*********************************************************************%

%*********************************************************************%
%*                       LOCAL DECLARATIONS                          *%
%*********************************************************************%
REAL
    EXPR_1
    ;
DOUBLE
      EXPR_2
      ;
STRING
      DATA_SET_NAME
      ,FIRST_1
      ,PRIOR_1
      ,NEXT_1
      ,P_AUDIT
      ,OPEN_TYPE
      ,BEGIN_1
      ,CONDITION_1
      ,SPACE_1
      ,DATA_BUFFER
      ,DATA_REQUEST
      ,ITEM_NAME
      ,EXPR_3
      ,EXCEPTIONMSG
      ;
BOOLEAN
      RSLT
      ,EXPR_4
      ;
```

```
%*********************************************************************%
%*             INTERNAL PROCEDURE TO HANDLE EXCEPTIONS               *%
%*********************************************************************%

PROCEDURE EXCEPTIONHANDLER;
BEGIN
STRING
    EXCEPTIONMSG
    ;
BOOLEAN
    STATUSRSLT
    ;

    STATUSRSLT:= DBSTATUS ;
    DBEXCEPTIONNAME(STATUSRSLT,EXCEPTIONMSG);
    WRITE(REMOTEFILE,<"EXCEPTION-NAME=",A14>,EXCEPTIONMSG);
    DBEXCEPTIONTEXT(STATUSRSLT,EXCEPTIONMSG);
    WRITE(REMOTEFILE,<"EXCEPTION-TEXT=">);
    WRITE(REMOTEFILE,<A60>,EXCEPTIONMSG);
END  OF EXCEPTIONHANDLER ;

%*********************************************************************%
%*                        BODY OF MAIN PROGRAM                       *%
%*********************************************************************%

      DMI.TITLE:= "DMINTERPRETER/EMPJOB";
%*********************************************************************%
      DATA_SET_NAME:="EMP";
      FIRST_1  :="FIRST";
      PRIOR_1  :="PRIOR";
      NEXT_1   :="NEXT";
      P_AUDIT  :="AUDIT";
      OPEN_TYPE:="UPDATE";
      BEGIN_1  :="BEGINNING";
      CONDITION_1 :="EMP-NO=11";
      SPACE_1     :=" ";

%*********************************************************************%
%*                      OPENING THE DATABASE                         *%
%*********************************************************************%

      RSLT:= DBOPEN(OPEN_TYPE) ;
      IF RSLT
         THEN  EXCEPTIONHANDLER;
      WRITE(REMOTEFILE,<"SUCCESSFUL OPEN OF EMPJOB">);

%*********************************************************************%
%*                        DELETING A RECORD                          *%
%*********************************************************************%

      RSLT:= DBBEGINTRANSACTION(P_AUDIT) ;
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;
```

```
      RSLT:= DBDELETE(FIRST_1,DATA_SET_NAME,CONDITION_1) ;
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;

      CONDITION_1:="EMP-NO=12" ;
      RSLT:= DBDELETE(FIRST_1,DATA_SET_NAME,CONDITION_1) ;
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;

      RSLT:= DBENDTRANSACTION(P_AUDIT) ;
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;

      WRITE(REMOTEFILE, <"SUCCESSFUL DELETE OF RECORD">);

%*********************************************************************%
%*                          CREATING A RECORD                        *%
%*********************************************************************%

      RSLT:= DBCREATE(DATA_SET_NAME,SPACE_1) ;
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;

      WRITE(REMOTEFILE, <"SUCCESSFUL CREATE OF RECORD">) ;

%*********************************************************************%
%*                        FILLING IN A RECORD                        *%
%*********************************************************************%
%
%        USING DBPUTREAL ENTRY POINT
%
      ITEM_NAME:= "EMP-NO";
      EXPR_1:=11;
      RSLT:= DBPUTREAL(DATA_SET_NAME,ITEM_NAME,EXPR_1);
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;

      ITEM_NAME:="EMP-SALARY";
      EXPR_1:=4500.95;
      RSLT:= DBPUTREAL(DATA_SET_NAME,ITEM_NAME, 2 * EXPR_1);
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;
%
%        USING DBPUTDOUBLE ENTRY POINT
%
      ITEM_NAME:= "EMP-SSN" ;
      EXPR_2:=573230911 ;
      RSLT:= DBPUTDOUBLE(DATA_SET_NAME,ITEM_NAME,EXPR_2 - 11 );
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;
```

```
%
%        USING DBDATA ENTRY POINT
%
      DATA_REQUEST:= "PUT REAL EMP EMP-DATE-HIRED" ;
      DATA_BUFFER:= "811223" ;
      RSLT:= DBDATA(DATA_REQUEST,DATA_BUFFER) ;
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;

      DATA_REQUEST:= "PUT STRING EMP EMP-LNAME" ;
      DATA_BUFFER:= "STACK" ;
      RSLT:= DBDATA(DATA_REQUEST,DATA_BUFFER) ;
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;

%
%        USING DBPUTSTRING ENTRY POINT
%
      ITEM_NAME:="EMP-FNAME" ;
      EXPR_3:="CATHY" ;
      RSLT:= DBPUTSTRING(DATA_SET_NAME,ITEM_NAME,EXPR_3 CAT "LINA" );
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;

%
%        USING DBPUTBOOLEAN
%
      ITEM_NAME:=  "EMP-SEC-CLEAR" ;
      EXPR_4:= TRUE ;
      RSLT:= DBPUTBOOLEAN(DATA_SET_NAME,ITEM_NAME,NOT EXPR_4);
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;

      WRITE(REMOTEFILE, <"SUCCESSFUL FILLING OF RECORD">) ;

%*********************************************************************%
%*                  RECORD SET-UP IN USER WORK AREA                  *%
%*********************************************************************%
%*                                                                   *%
%* EMP    EMP      EMP      EMP       EMP           EMP        EMP    *%
%*  NO   LNAME    FNAME     SSN    DATE-HIRED   SEC-CLEAR   SALARY    *%
%*-------------------------------------------------------------------*%
%*21   STACK   CATHYLINA 573230900   811223      FALSE      9001.9  *%
%*********************************************************************%

%*********************************************************************%
%*                         STORING A RECORD                          *%
%*********************************************************************%

      RSLT:= DBBEGINTRANSACTION(P_AUDIT) ;
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;
```

```
      RSLT:= DBSTORE(DATA_SET_NAME) ;
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;

      RSLT:= DBENDTRANSACTION(P_AUDIT) ;
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;

      WRITE(REMOTEFILE, <"SUCCESSFUL STORE OF RECORD">) ;

%*********************************************************************%
%*                    RE-CREATING A RECORD                           *%
%*********************************************************************%

      RSLT:= DBRECREATE(DATA_SET_NAME,SPACE_1) ;
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;

      WRITE(REMOTEFILE, <"SUCCESSFUL RECREATE OF RECORD">) ;

      ITEM_NAME:= "EMP-NO";
      EXPR_1:=12;
      RSLT:= DBPUTREAL(DATA_SET_NAME,ITEM_NAME,EXPR_1);
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;

      ITEM_NAME:= "EMP-SSN" ;
      EXPR_2:=549927328 ;
      RSLT:= DBPUTDOUBLE(DATA_SET_NAME,ITEM_NAME,EXPR_2 - 11 );
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;

      DATA_REQUEST:= "PUT REAL EMP EMP-DATE-HIRED" ;
      DATA_BUFFER:= "811228" ;
      RSLT:= DBDATA(DATA_REQUEST,DATA_BUFFER) ;
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;

      DATA_REQUEST:= "PUT STRING EMP EMP-LNAME" ;
      DATA_BUFFER:= "WOLVERTON" ;
      RSLT:= DBDATA(DATA_REQUEST,DATA_BUFFER) ;
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;

      ITEM_NAME:="EMP-FNAME" ;
      EXPR_3:="LAURA" ;
      RSLT:= DBPUTSTRING(DATA_SET_NAME,ITEM_NAME,EXPR_3 CAT "JANE" );
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;

      RSLT:= DBBEGINTRANSACTION(P_AUDIT) ;
      IF RSLT
```

```
      THEN   EXCEPTIONHANDLER  ;

   RSLT:= DBSTORE(DATA_SET_NAME) ;
   IF RSLT
      THEN   EXCEPTIONHANDLER  ;

   RSLT:= DBENDTRANSACTION(P_AUDIT) ;
   IF RSLT
      THEN   EXCEPTIONHANDLER  ;

   WRITE(REMOTEFILE, <"SUCCESSFUL STORE OF RECREATED RECORD">) ;

%*************************************************************************%
%*          SET-UP OF RE-CREATED RECORD IN USER WORK AREA                *%
%*************************************************************************%
%*                                                                       *%
%* EMP    EMP        EMP         EMP       EMP        EMP       EMP    *%
%* NO    LNAME      FNAME       SSN    DATE-HIRED  SEC-CLEAR  SALARY  *%
%*-----------------------------------------------------------------------*%
%* 12  WOLVERTON  LAURAJANE   549927317   811228      FALSE    9001.9  *%
%*************************************************************************%

%*************************************************************************%
%*                SETTING DATA SET TO THE BEGINNING                      *%
%*************************************************************************%

   RSLT:= DBSET(DATA_SET_NAME,BEGIN_1) ;
   IF RSLT
      THEN   EXCEPTIONHANDLER  ;

   WRITE(REMOTEFILE, <"SUCCESSFUL DBSET OF DATA SET">) ;

%*************************************************************************%
%*                        FINDING A RECORD                               *%
%*************************************************************************%

   RSLT:= DBFIND(FIRST_1,DATA_SET_NAME,CONDITION_1) ;
   IF RSLT
      THEN   EXCEPTIONHANDLER  ;

   WRITE(REMOTEFILE, <"SUCCESSFUL FIND OF RECORD">) ;

%*************************************************************************%
%*                        LOCKING A RECORD                               *%
%*************************************************************************%

   RSLT:= DBLOCK(FIRST_1,DATA_SET_NAME,CONDITION_1) ;
   IF RSLT
      THEN   EXCEPTIONHANDLER  ;

   WRITE(REMOTEFILE, <"SUCCESSFUL LOCK OF RECORD">) ;
```

```
%*********************************************************************%
%*                  VERIFYING ITEMS IN THE RECORD                    *%
%*********************************************************************%
%
%        USING DBGETREAL ENTRY POINT
%
      ITEM_NAME:="EMP-NO";
      RSLT:= DBGETREAL(DATA_SET_NAME,ITEM_NAME,EXPR_1);
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;
      WRITE(REMOTEFILE, <"EMP-NO=",I4>,EXPR_1) ;

      ITEM_NAME:="EMP_SALARY";
      RSLT:= DBGETREAL(DATA_SET_NAME,ITEM_NAME,EXPR_1);
      IF RSLT
         THEN
      WRITE(REMOTEFILE, <"EMP-SALARY=",F9.2>,EXPR_1);
%
%        USING DBGETDOUBLE ENTRY POINT
%
      ITEM_NAME:="EMP-SSN" ;
      RSLT:= DBGETDOUBLE(DATA_SET_NAME,ITEM_NAME,EXPR_2);
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;
      WRITE(REMOTEFILE, <"EMP-SSN=",I9>,EXPR_2) ;

%
%        USING DBDATA ENTRY POINT
%
      DATA_REQUEST:= "GET REAL EMP EMP-DATE-HIRED" ;
      RSLT:= DBDATA(DATA_REQUEST,DATA_BUFFER) ;
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;
      WRITE(REMOTEFILE, <"EMP-DATE-HIRED=",A6>,DATA_BUFFER) ;

      DATA_REQUEST:= "GET REAL EMP EMP-SALARY" ;
      RSLT:= DBDATA(DATA_REQUEST,DATA_BUFFER) ;
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;
      WRITE(REMOTEFILE, <"EMP-SALARY=",A6>,DATA_BUFFER) ;

      DATA_REQUEST:= "GET STRING EMP EMP-LNAME" ;
      RSLT:= DBDATA(DATA_REQUEST,DATA_BUFFER) ;
      IF RSLT
         THEN   EXCEPTIONHANDLER  ;
      WRITE(REMOTEFILE, <"EMP-LNAME=",A10>,DATA_BUFFER) ;

%
%        USING DBGETSTRING ENTRY POINT
%
      ITEM_NAME:="EMP-FNAME";
      RSLT:= DBGETSTRING(DATA_SET_NAME,ITEM_NAME,EXPR_3);
      IF RSLT
```

```
      THEN   EXCEPTIONHANDLER  ;
   WRITE(REMOTEFILE, <"EMP-FNAME=",A10>,EXPR_3);

%
%        USING DBGETBOOLEAN ENTRY POINT
%
   ITEM_NAME:="EMP-SEC-CLEAR" ;
   RSLT:= DBGETBOOLEAN(DATA_SET_NAME,ITEM_NAME,EXPR_4);
   IF RSLT
      THEN   EXCEPTIONHANDLER  ;
   WRITE(REMOTEFILE, <"EMP-SEC-CLEAR=",L5>,EXPR_4) ;

%*********************************************************************%
%*                        FREEING THE RECORD                         *%
%*********************************************************************%

   RSLT:= DBFREE(DATA_SET_NAME) ;
   IF RSLT
      THEN   EXCEPTIONHANDLER  ;

   WRITE(REMOTEFILE, <"SUCCESSFUL FREEING OF RECORD">) ;

%*********************************************************************%
%*                       CAUSING AN EXCEPTION                        *%
%*********************************************************************%
   WRITE(REMOTEFILE, <"CAUSING AN EXCEPTION">) ;

   RSLT:= DBSET(DATA_SET_NAME,BEGIN_1) ;
   IF RSLT
      THEN   EXCEPTIONHANDLER  ;

   RSLT:= DBFIND(PRIOR_1,DATA_SET_NAME,SPACE_1) ;
   IF RSLT
      THEN   EXCEPTIONHANDLER  ;

%
%*********************************************************************%
%*                           USING DBVERB                            *%
%*********************************************************************%
%
%         USING DBVERB ON A SET STATEMENT
%
   RSLT:= DBVERB("SET EMP TO BEGINNING");
   IF RSLT
      THEN  EXCEPTIONHANDLER ;
   WRITE(REMOTEFILE,<"SUCCESSFUL SET EMP TO BEGINNING">);
%
%         USING DBVERB ON A SEARCH STATEMENT
%
   RSLT:= DBVERB("FIND FIRST EMP AT EMP-NO=11");
   IF RSLT
      THEN EXCEPTIONHANDLER ;
   WRITE(REMOTEFILE,<"SUCCESSFUL FIND FIRST EMP AT EMP-NO=11">);
```

```
%
%         USING DBVERB ON A STATUS STATEMENT
%
      RSLT:= DBVERB("SET EMP TO BEGINNING");
      IF RSLT
         THEN EXCEPTIONHANDLER ;
      DBVERB("FIND PRIOR EMP ");
      RSLT:=DBVERB("DMSTATUS");  % TO FIND OUT IF THE FIND WAS OK
      DBEXCEPTIONNAME(RSLT,EXCEPTIONMSG);
      WRITE (REMOTEFILE,<"EXCEPTION-NAME=",A14>,EXCEPTIONMSG);
      DBEXCEPTIONTEXT(RSLT,EXCEPTIONMSG);
      WRITE(REMOTEFILE,<"EXCEPTION-TEXT=">);
      WRITE(REMOTEFILE,<A60>,EXCEPTIONMSG);
      WRITE(REMOTEFILE,<"SUCCESSFUL STATUS STATEMENT">);

%*********************************************************************%
%*                          CLOSING THE DATA BASE                     *%
%*********************************************************************%

      RSLT:= DBCLOSE ;
      IF RSLT
         THEN   EXCEPTIONHANDLER ;

      WRITE(REMOTEFILE, <"SUCCESSFUL CLOSE OF EMPJOB">) ;

%*********************************************************************%
%*********************************************************************%
END OF DMIEXAMPLE.
```

SAMPLE PL/I PROGRAM

This is a sample PL/I program.

```
/*********************************************************************/
/* In this program, the declaration of library entry points for PL/I  */
/* was inserted from the DATABASE/DMINTERPRETER symbolic file, because*/
/* the $ INCLUDE compiler control option is not available to  PL/I.   */
/*********************************************************************/

DMIEXAMPLE: PROC OPTIONS(MAIN, DOUBLE) ;

/*********************************************************************/
/*       P L / I   L I B R A R Y   D E C L A R A T I O N S           */
/*********************************************************************/

 DECLARE DMI LIBRARY;
 DECLARE DBOPEN ENTRY (CHARACTER(*))
    RETURNS (BIT(48))
    OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLOPEN');
 DECLARE DBCLOSE ENTRY
    RETURNS (BIT(48))
    OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLCLOSE');
 DECLARE DBBEGINTRANSACTION ENTRY (CHARACTER(*))
    RETURNS (BIT(48))
    OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLBEGINTRANSACTION');
 DECLARE DBENDTRANSACTION ENTRY (CHARACTER(*))
    RETURNS (BIT(48))
    OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLENDTRANSACTION');
 DECLARE DBCREATE ENTRY (CHARACTER(*), CHARACTER(*))
    RETURNS (BIT(48))
    OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLCREATE');
 DECLARE DBRECREATE ENTRY (CHARACTER(*), CHARACTER(*))
    RETURNS (BIT(48))
    OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLRECREATE');
 DECLARE DBSTORE ENTRY (CHARACTER(*))
    RETURNS (BIT(48))
    OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLSTORE');
 DECLARE DBFREE ENTRY (CHARACTER(*))
    RETURNS (BIT(48))
    OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLFREE');
 DECLARE DBSET ENTRY (CHARACTER(*), CHARACTER(*))
    RETURNS (BIT(48))
    OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLSET');
 DECLARE DBFIND ENTRY (CHARACTER(*), CHARACTER(*), CHARACTER(*))
    RETURNS (BIT(48))
    OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLFIND');
 DECLARE DBLOCK ENTRY (CHARACTER(*), CHARACTER(*), CHARACTER(*))
    RETURNS (BIT(48))
```

```
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLLOCK');
DECLARE DBDELETE ENTRY (CHARACTER(*), CHARACTER(*), CHARACTER(*))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLDELETE');
DECLARE DBVERB ENTRY (CHARACTER(*))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLVERB');
DECLARE DBGETCHARACTER ENTRY (CHARACTER(*), CHARACTER(*),
                              CHARACTER(*) VARYING)
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLGETSTRING');
DECLARE DBGETFLOAT ENTRY (CHARACTER(*), CHARACTER(*), BINARY FLOAT(38))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLGETREAL');

DECLARE DBGETDOUBLE ENTRY (CHARACTER(*), CHARACTER(*),
                           BINARY FLOAT(78))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLGETDOUBLE') ;

DECLARE DBGETBIT ENTRY (CHARACTER(*), CHARACTER(*), BIT(48))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLGETBOOLEAN');
DECLARE DBPUTCHARACTER ENTRY (CHARACTER(*), CHARACTER(*), CHARACTER(*))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLPUTSTRING');
DECLARE DBPUTFLOAT ENTRY (CHARACTER(*), CHARACTER(*), BINARY FLOAT(38))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLPUTREAL');

DECLARE DBPUTDOUBLE ENTRY (CHARACTER(*), CHARACTER(*),
                           BINARY FLOAT(78))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLPUTDOUBLE') ;

DECLARE DBPUTBIT ENTRY (CHARACTER(*), CHARACTER(*), BIT(48))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLPUTBOOLEAN');
DECLARE DBPUTNULL ENTRY (CHARACTER(*), CHARACTER(*))
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLPUTNULL');
DECLARE DBDATA ENTRY (CHARACTER(*), CHARACTER(*) VARYING)
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLDATA');
DECLARE DBSTATUS ENTRY
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLSTATUS');
DECLARE DBEXCEPTIONNAME ENTRY (BIT(48), CHARACTER(*) VARYING)
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLEXCEPTIONNAME');
DECLARE DBEXCEPTIONTEXT ENTRY (BIT(48), CHARACTER(*) VARYING)
   RETURNS (BIT(48))
   OPTIONS (LIBRARY = DMI, ACTUALNAME = 'ALGOLEXCEPTIONTEXT');
```

```
/*****************************************************************/
/*                    LOCAL DECLARATIONS                         */
/*****************************************************************/
DCL
      DMI_NAME          CHAR(24) INIT('DMINTERPRETER/EMPJOB'),
      DATA_SET_NAME   CHAR(17) INIT('EMP'),
      FIRST_1           CHAR( 5) INIT('FIRST'),
      PRIOR_1           CHAR( 5) INIT('PRIOR'),
      NEXT_1            CHAR( 4) INIT('NEXT'),
      P_AUDIT           CHAR( 5) INIT('AUDIT'),
      OPEN_TYPE         CHAR( 6) INIT('UPDATE'),
      BEGIN_1           CHAR( 9) INIT('BEGINNING'),
      CONDITION_1       CHAR( 9) INIT('EMP-NO=11'),
      SPACE_1           CHAR( 1) INIT(' '),
      DATA_BUFFER       CHAR(10) VAR,
      DATA_REQUEST      CHAR(40),
      RSLT                BIT(48),
      ITEM_NAME         CHAR(15) VAR,
      FLOAT_EXPR        FLOAT BINARY(38),
      DFLOAT_EXPR       FLOAT BINARY(78),
      STRING_EXPR       CHAR(10) VAR,
      BOOLEAN_EXPR      BIT(48),
      EXCEPTIONMSG        CHAR(132) VAR
      ;

/*****************************************************************/
/*            INTERNAL PROCEDURE TO HANDLE EXCEPTIONS            */
/*****************************************************************/
```

SAMPLE FORTRAN77 PROGRAM

The following is a sample FORTRAN77 program.

```
************************************************************************
* In this program the FORTRAN77 declaration of library entry points    *
* has been  included from the DATABASE/DMINTERPRETER symbolic file     *
* rather than explicitly declaring each entry point used.              *
************************************************************************


$ RESET FREE
*
************************************************************************
*            FORTRAN77 LIBRARY AND ENTRY POINT DECLARATIONS            *
************************************************************************

      BLOCK GLOBALS
         LIBRARY DMI(TITLE="DMINTERPRETER/EMPJOB.")
      END

%$ INCLUDE "DATABASE/DMINTERPRETER" 20200000-20249999

************************************************************************
*                         SUBROUTINE EXCEPT                            *
************************************************************************

      SUBROUTINE EXCEPT

%$ INCLUDE "DATABASE/DMINTERPRETER" 20290000-20299999

      CHARACTER      MSG        *70
      LOGICAL        RESULT

      RESULT = DBSTAT

      CALL DBEXCN(RESULT,MSG)
      WRITE(6,200) MSG

      CALL DBEXCT(RESULT,MSG)
      WRITE(6,200) MSG
  200 FORMAT(1X,A)
      RETURN
      END


%$ INCLUDE "DATABASE/DMINTERPRETER" 20290000-20299999
```

```
************************************************************************
*                      START OF MAIN PROGRAM                           *
************************************************************************

     CHARACTER DSNAME      *4      /"EMP"/,
    *          FIRST       *10      /"FIRST"/,
    *          PRIOR       *10      /"PRIOR"/,
    *          NEXT        *10      /"NEXT"/,
    *          BEGIN       *9      /"BEGINNING"/,
    *          AUDIT       *10      /"AUDIT"/,
    *          UPDATE      *6      /"UPDATE"/,
    *          SPACE       *1      /" "/,
    *          COND        *9      /"EMP-NO=11"/,
    *          DATREQ      *40     ,
    *          DATBUF      *22     ,
    *          ITMNAM      *13     ,
    *          MSG         *70     ,
    *          EXPR3       *22

     LOGICAL  RSLT,
    *         EXPR4

     DOUBLE PRECISION  EXPR2

     REAL   EXPR1


************************************************************************
*                        OPENING THE DATABASE                          *
************************************************************************

     RSLT = DBOPEN(UPDATE)
     IF (RSLT) CALL EXCEPT


************************************************************************
*                         DELETING A RECORD                            *
************************************************************************

     RSLT = DBBTR(AUDIT)
     IF (RSLT) CALL EXCEPT

     RSLT = DBDEL(FIRST,DSNAME,COND)
     IF (RSLT) CALL EXCEPT

     RSLT = DBETR(AUDIT)
     IF (RSLT) CALL EXCEPT
```

```
*************************************************************************
*                          CREATING A RECORD                            *
*************************************************************************

     RSLT = DBCR(DSNAME,SPACE)
     IF (RSLT) CALL EXCEPT
*************************************************************************
*                          FILLING IN A RECORD                          *
*************************************************************************

*
*         USING DBPUTREAL ENTRY POINT
*
     ITMNAM = "EMP-NO"
     EXPR1 = 11
     RSLT = DBPUTR(DSNAME,ITMNAM,EXPR1)
     IF (RSLT) CALL EXCEPT

     ITMNAM = "EMP-SALARY"
     EXPR1 = 4500.95
     RSLT = DBPUTR(DSNAME,ITMNAM, 2 * EXPR1)
     IF (RSLT) CALL EXCEPT

*
*        USING DBPUTDOUBLE ENTRY POINT
*
     ITMNAM = "EMP-SSN"
     EXPR2 = 573230911
     RSLT = DBPUTD(DSNAME,ITMNAM, EXPR2 - 11)
     IF (RSLT) CALL EXCEPT

*
*               USING DBDATA ENTRY POINT
*
     DATREQ = "PUT REAL EMP EMP-DATE-HIRED"
     DATBUF = "811223"
     RSLT = DBDATA(DATREQ,DATBUF)
     IF (RSLT) CALL EXCEPT

     DATREQ = "PUT CHARACTER EMP EMP-LNAME"
     DATBUF = "STACK"
     RSLT = DBDATA(DATREQ,DATBUF)
     IF (RSLT) CALL EXCEPT

*
*        USING DBPUTCHARACTER ENTRY POINT
*
     ITMNAM = "EMP-FNAME"
     EXPR3 = "CATHY"
     RSLT = DBPUTC(DSNAME,ITMNAM,EXPR3(1:5) // "LINA")
     IF (RSLT) CALL EXCEPT
```

```
*
*       USING DBPUTLOGICAL ENTRY POINT
*
      ITMNAM = "EMP-SEC-CLEAR"
      EXPR4 = .TRUE.
      RSLT = DBPUTL(DSNAME,ITMNAM,.NOT. EXPR4)
      IF (RSLT) CALL EXCEPT

*************************************************************************
*                   RECORD SET-UP IN USER WORK AREA                     *
*************************************************************************
*                                                                       *
*  EMP     EMP      EMP       EMP        EMP            EMP         EMP  *
*   NO    LNAME    FNAME      SSN     DATE-HIRED     SEC-CLEAR    SALARY  *
*-----------------------------------------------------------------------*
*22    STACK CATHYLINA 573230900    811223          FALSE     9001.9   *
*************************************************************************

*************************************************************************
*                           STORING A RECORD                            *
*************************************************************************

      RSLT = DBBTR(AUDIT)
      IF (RSLT) CALL EXCEPT

      RSLT = DBSTORE(DSNAME)
      IF (RSLT) CALL EXCEPT

      RSLT = DBETR(AUDIT)
      IF (RSLT) CALL EXCEPT

*************************************************************************
*                        RE-CREATING A RECORD                           *
*************************************************************************

      RSLT = DBRECR(DSNAME,SPACE)
      IF (RSLT) CALL EXCEPT

      ITMNAM = "EMP-NO"
      EXPR1 = 12
      RSLT = DBPUTR(DSNAME,ITMNAM,EXPR1)
      IF (RSLT) CALL EXCEPT

      ITMNAM = "EMP-SSN"
      EXPR2 = 549273281
      RSLT = DBPUTD(DSNAME,ITMNAM, EXPR2 - 12)
      IF (RSLT) CALL EXCEPT

      DATREQ = "PUT STRING EMP EMP-LNAME"
      DATBUF = "BLACK"
      RSLT = DBDATA(DATREQ,DATBUF)
      IF (RSLT) CALL EXCEPT
```

```
      ITMNAM = "EMP-FNAME"
      EXPR3 = "SARAH"
      RSLT = DBPUTC(DSNAME,ITMNAM,EXPR3(1:5) // "LEE")
      IF (RSLT) CALL EXCEPT

      RSLT = DBBTR(AUDIT)
      IF (RSLT) CALL EXCEPT

      RSLT = DBSTORE(DSNAME)
      IF (RSLT) CALL EXCEPT

      RSLT = DBETR(AUDIT)
      IF (RSLT) CALL EXCEPT

*************************************************************************
*                  SETTING DATA SET TO THE BEGINNING                    *
*************************************************************************

      RSLT = DBSET(DSNAME,BEGIN)
      IF (RSLT) CALL EXCEPT

*************************************************************************
*                          FINDING A RECORD                             *
*************************************************************************

      RSLT = DBFIND(FIRST,DSNAME,COND)
      IF (RSLT) CALL EXCEPT

*************************************************************************
*                         LOCKING A RECORD                              *
*************************************************************************

      RSLT = DBLOCK(FIRST,DSNAME,COND)
      IF (RSLT) CALL EXCEPT

*************************************************************************
*                    VERIFYING ITEMS IN A RECORD                        *
*************************************************************************
*
*
*        USING DBGETREAL ENTRY POINT
*
      ITMNAM = "EMP-NO"
      RSLT = DBGETR(DSNAME,ITMNAM,EXPR1)
      IF (RSLT) CALL EXCEPT

      ITMNAM = "EMP-SALARY"
      RSLT = DBGETR(DSNAME,ITMNAM,EXPR1)
      IF (RSLT) CALL EXCEPT
*
*
*        USING DBGETDOUBLE ENTRY POINT
*
```

```
      ITMNAM = "EMP-SSN"
      RSLT = DBGETD(DSNAME,ITMNAM,EXPR2)
      IF (RSLT) CALL EXCEPT
*
*
*        USING DBDATA ENTRY POINT
*
      DATREQ = "GET REAL EMP EMP-DATE-HIRED"
      RSLT = DBDATA(DATREQ,DATBUF)
      IF (RSLT) CALL EXCEPT

      DATREQ = "GET CHARACTER EMP EMP-LNAME"
      RSLT = DBDATA(DATREQ,DATBUF)
      IF (RSLT) CALL EXCEPT
*
*
*        USING DBGETCHARACTER ENTRY POINT
*
      ITMNAM = "EMP-FNAME"
      RSLT = DBGETC(DSNAME,ITMNAM,EXPR3)
      IF (RSLT) CALL EXCEPT
*
*
*        USING DBGETLOGICAL ENTRY POINT
*
      ITMNAM = "EMP-SEC-CLEAR"
      RSLT = DBGETL(DSNAME,ITMNAM,EXPR4)
      IF (RSLT) CALL EXCEPT

*************************************************************************
*                          FREEING THE RECORD                           *
*************************************************************************

      RSLT = DBFREE(DSNAME)
      IF (RSLT) CALL EXCEPT

*************************************************************************
*                       CAUSING AN EXCEPTION                            *
*************************************************************************

      RSLT = DBSET(DSNAME,BEGIN)
      IF (RSLT) CALL EXCEPT

      RSLT = DBFIND(PRIOR,DSNAME,SPACE)
      IF (RSLT) CALL EXCEPT
```

```
*************************************************************************
*                            USING DBVERB                               *
*************************************************************************
*
*         USING DBVERB ON A SET STATEMENT
*
      RSLT = DBVERB("SET EMP TO BEGINNING")
      IF (RSLT) CALL EXCEPT
*
*         USING DBVERB ON A SEARCH STATEMENT
*
      RSLT = DBVERB("FIND FIRST EMP AT EMP-NO=11")
      IF (RSLT) CALL EXCEPT
*
*         USING DBVERB ON A STATUS STATEMENT
*
      RSLT = DBVERB("SET EMP TO BEGINNING")
      IF (RSLT) CALL EXCEPT
      RSLT = DBVERB("FIND PRIOR EMP")
      RSLT = DBVERB("DMSTATUS")
      CALL DBEXCN(RSLT,MSG)
      CALL DBEXCT(RSLT,MSG)

*************************************************************************
*                         CLOSING THE DATA BASE                         *
*************************************************************************

      RSLT = DBCLOSE
      IF (RSLT) CALL EXCEPT

      END
```

GLOSSARY

ABSN

See "Audit Block Serial Number."

access

A logical index structure that defines the physical ordering of records in direct, ordered, and random data sets. An access functions like a set, but no physical file is associated with an access.

Accessroutines

Routines that perform all physical and logical management of a data base and allow many users to access the data base concurrently. Each data management statement that a user language program executes invokes a portion of the Accessroutines to perform all file management functions that the statement requires.

ACTUALNAME clause

A clause that allows the "renaming" of a library entry point within a user program by using a different procedure identifier to link to the original library entry point name. This clause is used in ALGOL, PL/I, and FORTRAN77 application programs when a procedure is declared to be an entry point of a particular library.

alpha item

A data item that stores alphanumeric information (letters, numbers, special characters, and blanks) as EBCDIC characters. An alpha item cannot be used in calculations.

argument

In PL/I, an expression, a constant, a variable, or a computational built-in function. When a PL/I program invokes an entry point of a library, a relationship is established between the arguments of the invoking statement and the parameters of the invoked entry point. A parameter itself can be passed as an argument.

Audit Block Serial Number (ABSN)

A number associated with a block in the audit file. The first block in the first audit file is noted as ABSN 0. The ABSN is incremented by one through the last audit file.

audited data base

A data base that stores a record of changes (called the "audit trail"), which can be used for data base recovery in the event of hardware or software failure.

automatic subset

A subset declared with a condition that specifies which members of the data set are to be included in the subset. Entries are automatically inserted into or removed from the subset when records are added to or deleted from the data set.

binary search

A quick method of searching for records that are in a known sequence by successively halving the area to be searched.

block

A group of physically adjacent records packaged together so that they can be transferred to or from a physical device as a group.

Boolean

An adjective that describes variables having a value of TRUE or FALSE.

Boolean item

A data item that stores information having a logical value of either TRUE or FALSE.

CALL statement

In COBOL, the "CALL nonnumeric literal" statement is the only construct that allows a program to call a library entry point. The CALL statement can pass required parameters in an optional USING clause, and receive a result in an optional GIVING clause.

CANDE

See "Command AND Edit."

CHANGE ATTRIBUTE statement

In COBOL, the "CHANGE ATTRIBUTE" statement is the only construct that can be used to change library attributes. When programming for the interpretive interface, this statement is used to change the TITLE attribute of the library to the actual name of the library code file.

Command AND Edit (CANDE)

A Unisys Message Control System (MCS) that provides generalized file preparation and updating capabilities and task control in an interactive, terminal-oriented environment.

complex key

A key composed of multiple key items.

current record

The actual data set record that a program is currently referencing.

DASDL

See "Data and Structure Definition Language."

Data and Structure Definition Language (DASDL)

The language used to describe a data base logically and physically, and to specify criteria to ensure the integrity of data stored in the data base. DASDL is the source language that is input to the DASDL compiler, which creates or updates the data base description file from the input.

data base

An integrated, centralized system of data files and program utilities designed to support an application. The data sets and associated index structures are defined by a single description. Ideally, all the permanent data pertinent to a particular application resides in a single data base.

data base definition

A description of the logical and physical structures of a data base.

data independence

The property of being able to change the structural format of a data base without requiring changes to or recompilation of unaffected application programs that use the data base.

data item

A field in a data base record or transaction format that contains a particular type of information.

Data Management System II (DMSII)

A specialized system software package used to describe a data base and maintain the relationships between the data elements in the data base.

data set

A collection of related data records stored in a file on a random-access storage device. A data set is similar to a conventional file. It contains data items and has logical and physical properties similar to files. However, unlike conventional files, data sets can contain other data sets, sets, and subsets.

The data in a data set is formatted in a special manner so that it can be accessed by DMSII software. Application programs are expected to access a data set through the DMSII software rather than directly as files.

data set record

A record contained in a data set, which can be accessed through a DMSII statement.

data transfer entry point

A type of entry point peculiar to the DMINTERPRETER library whereby an item needed from a data set record is individually moved to or from a variable or expression that has been declared in the application program.

data type

An interpretation applied to a string of bits. Data types can be classified as "structured" or "scalar." Structured data types are collections of individual data items of the same or different data types. Scalar data types include real, integer, double precision, complex, logical (also called Boolean), character, pointer, and label. Most programming languages provide a declaration statement or a standard convention to indicate the data type of a variable.

DECLARE statement

In PL/I, a statement used in an explicit and contextual declaration to establish a given identifier as a name that is valid throughout a specified scope in a program. When programming for the interpretive interface, the DECLARE statement is used to explicitly declare an identifier that represents the DMINTERPRETER library, and to specify a value for the TITLE attribute associated with the library.

description file

The file produced by the Data and Structure Definition Language (DASDL) or Transaction Formatting Language (TFL) compiler that contains information used when compiling all tailored software and all DMSII user language programs for a particular data base or transaction base.

directory

A file with the layout for each field of the record that it describes. A directory describes the layout of records within a file.

disjoint

The condition of a data set, set, or subset when it is not contained in another data set. "Disjoint" is the opposite of "embedded."

DMSII

See "Data Management System II."

EBCDIC

Extended Binary Coded Decimal Interchange Code. An 8-bit code representing 256 graphic and control characters that are the native character set of Unisys A Series and B 5900, B 6900, and B 7900 systems.

embedded

A data set, set, or subset contained within another data set. "Embedded" is the opposite of "disjoint."

entity

An item about which information is stored. An entity can be tangible or intangible. An entity is further defined by attributes, the characteristics of the entities.

entry point

A procedure in a library program that can be called by another program.

exception

An error result returned to an application program by the DMSII Accessroutines explaining why a requested data base operation was not performed.

GIVING clause

In COBOL, an optional clause in the "CALL non-numeric literal" statement that specifies a variable as the destination for the integer value returned from the procedure call.

global data item

A data item, group item, or population item that is not part of any data set. Global data items generally contain information such as control totals, hash totals, and populations applying to the entire data base.

group item

A data item that contains a collection of subordinate data items and group items.

IN LIBRARY statement

In FORTRAN77, the statement that associates an entry point's logical expression name with the library that is known to the application program through the library identifier.

INCLUDE

A compiler control option that allows the user to insert an entire file, or sequence range within a file, into a program.

indexed sequential data set

A data set organized to combine the efficiency of sequential organization with the ability to rapidly access records out of sequence. Used only on direct-access devices.

input/output mapping clause

A clause that can be used with the FIND, DELETE, GET, PUT, LOCK, and STORE data management statements for assigning the value of a data base item to a program variable (input) and/or assigning the value of a program variable to a data base item (output).

integer

A whole number.

internal file name

The name used to declare a file in a program. The internal name of a file is given by the value of its INTNAME file attribute. The INTNAME file attribute can then be used in Work Flow Language (WFL) file equations.

INTNAME

See "internal file name."

intrinsic

A program routine for common mathematical and other operations that is loaded onto the system separately. An intrinsic can be invoked by the Master Control Program (MCP), by user programs, or by the system operator.

intrinsic data set

A data set created by system designers as part of a Data Management System II (DMSII) software package, as opposed to a data set created by a user.

Kanji

The standard Japanese character set for information exchange. Each Kanji character is 2 bytes (16 bits) in length and takes two positions on a form image.

key

A field used to locate or identify a record.

key item

A data item or group item that serves as a retrieval key for a set, subset, or access.

library

A collection of one or more named routines or entry points that are stored in a file and can be called by other programs.

library identifier

An internal name within an application program that represents the library code file being accessed.

linear search

A method for searching a list record by record.

link item

A data base item that enables one data set record to refer to another.

logical data base

A collection of structures declared in the Data and Structure Definition Language (DASDL) that provide a view of the data base, enforce structure-level security, and achieve data independence. When a logical data base is declared in DASDL, the data sets, sets, subsets, and remaps to be included in it are listed.

Programs can use only the structures contained in the logical data base that they invoke. Through guard files that limit use of logical data bases and use of the physical data base, the data base administrator can control which structures any program accesses.

logical expression

In FORTRAN77, a rule for computing a value corresponding to ".TRUE." or ".FALSE.". Logical expressions consist of any valid combination of logical operands, logical operators, and parentheses. Each FORTRAN77 entry point of the DMINTERPRETER library is a logical expression.

logical function

A FORTRAN function that returns a value of ".TRUE." or ".FALSE.".

major key

The first key in a complex key.

manual subset

A subset that has no condition specifying which data set records are to be included in the subset. The user must add and delete manual subset entries, using the INSERT and REMOVE statements.

master

A data set or record that contains one or more embedded data sets or records. "Master," "parent," and "owner" are synonymous.

minor key

Any key in a complex key that is not the major (first) key.

nonkey item

An item in the data set being indexed that is not a key item in the selected set.

null

The value contained in an item that does not contain valid information.

numeric item

A data item whose description restricts its contents to a value represented by characters chosen from the digits 0 (zero) through 9. If signed, the item can also contain a "+", "-", or other representation of an operational sign.

OCCURS clause

In a Data and Structure Definition Language (DASDL) data item declaration, a clause that establishes an ordered (subscripted) collection of data items with identical attributes.

owner

See "master."

parent

A process that owns the critical block of a dependent process. If the parent exits the critical block before the dependent process terminates, the dependent process is discontinued.

path

A specific location within the logical ordering of a data set, set, subset, or access.

physical data base

An entire data base as it is stored on a particular recording medium such as disk. Whereas a logical data base represents only parts of a data base to be used for limited purposes, a physical data base is an entire data base.

population

For disjoint data sets, the number of records in the data set. For embedded data sets, the population is the number of records in the embedded data set owned by the current master.

railroad diagram

A graph depicting the allowable combinations of elements that form the proper syntactic constructs of a language.

real item

Data items that store signed or unsigned, fractional or whole values in single-precision, floating-point form.

real number

Any number, including fractions and whole numbers.

record

A group of logically related items of data within a file that are treated as a unit by the input/output (I/O) subsystem.

remap

A logical data record that redefines a physical data set record by omitting, reordering, or renaming the items.

remote file

A file with a KIND attribute specified as REMOTE. A remote file enables object programs to communicate interactively with a data communications station. Any program that involves data entry or inquiry to update records uses either remote files or the direct programmatic interface to the Communications Management System (COMS) to allow one or more stations to update the record.

restart data set (RDS)

A data set containing restart records that application programs can access to recover data base information after a system failure.

result descriptor

A 48-bit word that is the standard Data Management System II (DMSII) exception word. A result descriptor includes the exception category and subcategory, and the structure number involved. When an exception occurs, the last bit in the 48-bit word is set to 1. When an operation is successful, the 48-bit word contains all zeros.

selection expression

The entire complement of selection criteria used in a FIND, LOCK, or DELETE statement to locate a data set record. The definition of selection expression encompasses both the select options (FIRST, NEXT, LAST, and PRIOR), and all the variations for the key condition.

Semantic Information Manager (SIM)

The basis of the InfoExec (tm)* environment. SIM is a data base management system used to describe and maintain associations among data by means of subclass-superclass relationships and linking attributes.

set

A file of indexes that refers to all the records of a single data set. Sets are automatically maintained by the system. Sets permit access to the records of a data set in some logical sequence and are normally used to optimize certain types of retrievals of the data set records.

SIM

See "Semantic Information Manager."

*InfoExec is a trademark of Unisys Corporation.

string

A connected sequence or group of characters.

structure

Any physical entity in a data base; particularly, a data set, set, subset, access, or remap.

subscript

A number that is an index into an array.

subset

An index structure that is identical to a set, except that it need not contain a record for every record of the data set. A set must index every record in its associated data set, whereas a subset can index zero, one, several, or all the data set records. A subset might or might not be automatically maintained by Data Management System II (DMSII).

symbolic name

A basic syntactic item of the FORTRAN77 language. A symbolic name is a sequence of one to six letters or digits, the first of which must be a letter.

syncpoint

A point in time when no program updating a data base is in a transaction state, and a syncpoint record is written to the data base audit file.

syntax

The rules or grammar of a language.

TFL

See "Transaction Formatting Language."

TITLE

A file attribute whose value is the external name of a file. By default, this value is the value of the INTNAME attribute. For a file whose KIND attribute is equal to DISK or PACK, TITLE can be assigned a value of the form "<file name> ON <family name>"; thus, the values of the TITLE and FILENAME attributes, both of which specify the external file name, can be different.

Transaction Formatting Language (TFL)

The Unisys language used to write source files that are compiled to produce description files for transaction bases.

transaction state

The period in a user language program between a BEGINTRANSACTION statement and an ENDTRANSACTION statement. Programs making changes to an audited data base are required to be in transaction state when the changes are made.

unordered

An adjective describing data sets, sets, and subsets that are not maintained in a user-specified order.

update

To modify the information in a data base or transaction base.

user language interface statements

Unisys data management statements that have been implemented in COBOL, COBOL74, ALGOL, RPG, and PL/I to allow programs written in these languages to invoke a data base, manipulate data, and handle exception categories.

user work area

A memory area in a user program where data records are constructed, accessed, or modified. The Accessroutines maintain one user work area for each data set or remap invoked by a program.

USING clause

In COBOL, an optional clause in the "CALL non-numeric literal" statement that identifies the parameters being passed to a library entry point.

variable

An object whose value can be changed during program execution.

variable format

A record that consists of two parts: a fixed part and a variable-format part. A single record description exists for the fixed part. The variable-format part can describe several variable parts. An individual record is constructed by using the fixed part alone, or by joining the fixed part with one of the variable parts.

variable-format data set

A data set with records that can have several different formats. All records in such a data set, known as variable-format records, consist of a common fixed part and optionally include one of the variable parts declared in the Data and Structure Definition Language (DASDL).

word

A unit of computer memory. On A Series and B 5900, B 6900, and B 7900 systems, a word consists of 48 bits used for storage plus tag bits used to indicate how the word is interpreted.

BIBLIOGRAPHY

A Series ALGOL Programming Reference Manual, Volume 1: Basic Implementation (form 1169844). Unisys Corporation.

A Series COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation (form 1169877). Unisys Corporation.

A Series COBOL Programming Reference Manual (form 1169786). Unisys Corporation.

A Series DMSII User Language Interface Programming Guide (form 1180536). Unisys Corporation.

A Series DMSII Utilities Operations Guide (form 1163839). Unisys Corporation.

A Series FORTRAN77 Programming Reference Manual (form 1222666). Unisys Corporation.

A Series PL/I Reference Manual (form 1169620). Unisys Corporation.

INTERPRETIVE INTERFACE PROGRAMMING REFERENCE MANUAL

PCN 1163813-003 Release 3.8 May 1989

NOTES

ACTUALNAME clause, 100, 187, 269
ALGOL sample programs, See Sample programs
ALGOLBEGINTRANSACTION, 106, 192
ALGOLCLOSE, 105, 191
ALGOLCREATE, 110, 196
ALGOLDATA, 170, 253
ALGOLDELETE, 129, 213
ALGOLENDTRANSACTION, 108, 194
ALGOLEXCEPTIONNAME, 177, 259
ALGOLEXCEPTIONTEXT, 175, 257
ALGOLFIND, 121, 206
ALGOLFREE, 116, 202
ALGOLGETBOOLEAN, 230
ALGOLGETDOUBLE, 150, 233
ALGOLGETREAL, 144, 227
ALGOLGETSTRING, 139, 222
ALGOLGGETBOOLEAN, 147
ALGOLLOCK, 125, 209
ALGOLOPEN, 103, 189
ALGOLPUTBOOLEAN, 165, 248
ALGOLPUTDOUBLE, 245
ALGOLPUTNULL, 168, 251
ALGOLPUTREAL, 159, 242
ALGOLPUTSTRING, 153, 236
ALGOLRECREATE, 112, 198
ALGOLSET, 118, 204
ALGOLSETLIMIT, 179, 261
ALGOLSTATUS, 174, 256
ALGOLSTORE, 114, 200
ALGOLVERB, 133, 216
ALOGOLPUTDOUBLE, 162
ALPHA item, 349
   as in ALGOLGETKANJI, 142, 225
   as in ALGOLGETSTRING, 139, 222
   as in ALGOLPUTKANJI, 156, 239
   as in DBGETDISPLAY, 61
   as in DBGETKANJI, 63
   as in DBPUTKANJI, 74
   as in FORTRAN77GETCHARACTER, 303
   as in FORTRAN77GETKANJI, 306
   as in FORTRAN77PUTKANJI, 320
<arithmetic expression>
   as in ALGOLPUTDOUBLE, 162, 245
   as in ALGOLPUTREAL, 160, 242
   as in FORTRAN77PUTDOUBLE, 326
   as in FORTRAN77PUTREAL, 323
Attribute-setting entry points, 10
   in ALGOL, 179
   in COBOL/COBOL74, 92
   in FORTRAN77, 343
   in PL/I, 261
<audit>, See Audit parameter
   as in ALGOLENDTRANSACTION, 108, 194

<audit> (cont.)
as in DBENDTRANSACTION, 34
as in FORTRAN77ENDTRANSACTION, 276
Audit parameter
as in ALGOLBEGINTRANSACTION, 106, 192
as in DBBEGINTRANSACTION, 32
as in FORTRAN77BEGINTRANSACTION, 274
<auditsync>
as in ALGOLENDTRANSACTION, 108, 194
as in DBENDTRANSACTION, 34
as in FORTRAN77ENDTRANSACTION, 276

BINARY, 349
<bit string expression>
as in ALGOLPUTBOOLEAN, 248
BOOLEAN data item, 349
<Boolean expression>
as in ALGOLPUTBOOLEAN, 165
BOOLEAN value, 84, 171, 253
BUILDINQ, 15

Call non-numeric literal statement, 27
syntax, 27
CHARACTER data item
as in FORTRAN77DATA, 335
CHARACTER EXPRESSION
as in FORTRAN77PUTCHARACTER, 317
as in FORTRAN77PUTKANJI, 320
COBOL sample programs, See Sample programs
<condition>
as in ALGOLDELETE, 129, 213
as in ALGOLFIND, 121, 206
as in ALGOLLOCK, 125, 209
as in DBDELETE, 52
as in DBFIND, 46
as in DBLOCK, 49
as in FORTRAN77DELETE, 294
as in FORTRAN77FIND, 288
as in FORTRAN77LOCK, 291
COUNT data item, 349
as in ALGOLGETDOUBLE, 150, 233
as in ALGOLGETREAL, 144, 227
as in DBGETDOUBLE, 69
as in DBGETREAL, 65
as in FORTRAN77GETDOUBLE, 314
as in FORTRAN77GETREAL, 308
<create statement>
as in ALGOLVERB, 135, 216
as in DBVERB, 57
as in FORTRAN77VERB, 297

CURRENT
   as in DBDELETE, 53
   as in DBFIND, 47

<data>
   as in ALGOLDATA, 170, 253
   as in DBDATA, 84
   as in FORTRAN77DATA, 334
DATA, 351
<data base>
   as in ALGOLFIND, 208
   as in ALGOLFREE, 116, 202
   as in ALGOLGETBOOLEAN, 148, 231
   as in ALGOLGETDOUBLE, 151, 234
   as in ALGOLGETKANJI, 142, 225
   as in ALGOLGETREAL, 145, 228
   as in ALGOLGETSTRING, 140, 223
   as in ALGOLLOCK, 127, 211
   as in ALGOLPOUTDOUBLE, 163
   as in ALGOLPUTBOOLEAN, 166, 249
   as in ALGOLPUTDOUBLE, 246
   as in ALGOLPUTKANJI, 157, 240
   as in ALGOLPUTNULL, 169, 252
   as in ALGOLPUTREAL, 160, 243
   as in ALGOLPUTSTRING, 154, 237
   as in ALGOLSTORE, 114, 200
   as in DBFIND, 48
   as in DBFREE, 42
   as in DBGETBOOLEAN, 68
   as in DBGETDISPLAY, 62
   as in DBGETDOUBLE, 70
   as in DBGETKANJI, 63
   as in DBGETREAL, 66
   as in DBLOCK, 51
   as in DBPUTBOOLEAN, 79
   as in DBPUTDISPLAY, 73
   as in DBPUTDOUBLE, 81
   as in DBPUTKANJI, 75
   as in DBPUTNULL, 82
   as in DBPUTREAL, 77
   as in DBSTORE, 40
   as in FORTRAN77FIND, 290
   as in FORTRAN77FREE, 284
   as in FORTRAN77GETCHARACTER, 304
   as in FORTRAN77GETDOUBLE, 315
   as in FORTRAN77GETKANJI, 306
   as in FORTRAN77GETLOGICAL, 312
   as in FORTRAN77GETREAL, 309
   as in FORTRAN77LOCK, 293
   as in FORTRAN77PUTCHARACTER, 318
   as in FORTRAN77PUTDOUBLE, 327
   as in FORTRAN77PUTKANJI, 321

<data base> (cont.)
as in FORTRAN77PUTLOGICAL, 330
as in FORTRAN77PUTNULL, 333
as in FORTRAN77PUTREAL, 324
as in FORTRAN77STORE, 282
<data base statement>
as in ALGOLVERB, 135, 216
as in DBVERB, 57
as in FORTRAN77VERB, 297
Data base structure, determining, 345
<data request>
as in ALGOLDATA, 170, 253
as in DBDATA, 84
as in FORTRAN77DATA, 334
<data set>, 36, 38
as in ALGOLCREATE, 110, 196
as in ALGOLDELETE, 129, 213
as in ALGOLFIND, 122, 207
as in ALGOLFREE, 116, 202
as in ALGOLGETBOOLEAN, 147, 231
as in ALGOLGETDOUBLE, 151, 234
as in ALGOLGETKANJI, 142, 225
as in ALGOLGETREAL, 145, 228
as in ALGOLGETSTRING, 140, 223
as in ALGOLLOCK, 126, 210
as in ALGOLPUTBOOLEAN, 166, 249
as in ALGOLPUTDOUBLE, 163, 246
as in ALGOLPUTKANJI, 156, 240
as in ALGOLPUTNULL, 169, 252
as in ALGOLPUTREAL, 160, 243
as in ALGOLPUTSTRING, 154, 237
as in ALGOLRECREATE, 112, 198
as in ALGOLSTORE, 114, 200
as in DBDELETE, 52
as in DBFIND, 47
as in DBFREE, 42
as in DBGETBOOLEAN, 68
as in DBGETDISPLAY, 61
as in DBGETDOUBLE, 70
as in DBGETKANJI, 63
as in DBGETREAL, 66
as in DBLOCK, 50
as in DBPUTBOOLEAN, 78
as in DBPUTDISPLAY, 73
as in DBPUTDOUBLE, 81
as in DBPUTKANJI, 74
as in DBPUTNULL, 82
as in DBPUTREAL, 77
as in DBSTORE, 40
as in FORTRAN77CREATE, 278
as in FORTRAN77DELETE, 294
as in FORTRAN77FIND, 289
as in FORTRAN77FREE, 284

<data set> (cont.)
as in FORTRAN77GETCHARACTER, 304
as in FORTRAN77GETDOUBLE, 315
as in FORTRAN77GETKANJI, 306
as in FORTRAN77GETLOGICAL, 312
as in FORTRAN77GETREAL, 309
as in FORTRAN77LOCK, 292
as in FORTRAN77PUTCHARACTER, 318
as in FORTRAN77PUTDOUBLE, 327
as in FORTRAN77PUTKANJI, 321
as in FORTRAN77PUTLOGICAL, 330
as in FORTRAN77PUTNULL, 333
as in FORTRAN77PUTREAL, 324
as in FORTRAN77RECREATE, 280
as in FORTRAN77STORE, 282
Data transfer entry points, 7
in ALGOL, 138
in COBOL/COBOL74, 60
in FORTRAN77, 302
in PL/I, 221
DATABASE : <data base name> card, 19
DATASET, 348
DATASET-SUBTYPE, 349
DB-ID, 348
DB-NAME, 348
DB-OWNER, 348
DB-TYPE, 348
DBBEGINTRANSACTION, 32
DBCLOSE, 31
DBCREATE, 36
DBDATA, 84
DBDELETE, 52
DBENDTRANSACTION, 34
DBEXCEPTIONNAME, 90
DBEXCEPTIONTEXT, 88
DBFIND, 46
DBFIND enhancements, 389
DBFREE, 42
DBGETBOOLEAN, 67
DBGETDISPLAY, 61
DBGETDOUBLE, 69
DBGETKANJI, 63
DBGETREAL, 65
DBLOCK, 49
DBOPEN, 29
DBPUTBOOLEAN, 78
DBPUTDISPLAY, 72
DBPUTDOUBLE, 80
DBPUTKANJI, 74
DBPUTNULL, 82
DBPUTREAL, 76
DBRECREATE, 38
DBSET, 44

DBSETLIMIT, 92
DBSTATUS, 87
DBSTORE, 40
DBSTRUCTURE items, 348
DBVERB, 55
DIRECT, 349
<direction>
   as in ALGOLDELETE, 129, 213
   as in ALGOLFIND, 121, 206
   as in ALGOLLOCK, 125, 209
   as in DBDELETE, 52
   as in DBFIND, 46
   as in DBLOCK, 49
   as in FORTRAN77DELETE, 294
   as in FORTRAN77FIND, 288
   as in FORTRAN77LOCK, 291
DMINTERPRETER card, 19
DMINTERPRETER files, 13
DMINTERPRETER generation, 15
   from job decks, 19
   from remote terminals, 15
   Remote, using the NOZIP option, 18
DMINTERPRETER library, 5
DMSUPPORT library, 3, 86, 173, 255, 337, 338
   as in ALGOLEXCEPTIONNAME exception category, 177, 259
   as in ALGOLEXCEPTIONTEXT exception message, 175, 257
   as in ALGOLSTATUS exception result, 174, 256
   as in DBEXCEPTIONNAME exception category, 9, 90
   as in DBEXCEPTIONTEXT exception message, 88
   as in DBSTATUS exception word, 87
   as in FORTRAN77EXCEPTIONNAME exception category, 341
   as in FORTRAN77EXCEPTIONTEXT exception message, 339

EBCDIC string, 97, 100, 351
Entry point declaration
   in ALGOL, 99
   in FORTRAN77, 267
<entry point name>
   as in PL/I, 186
<entry point options list>
   as in PL/I, 187
Entry points, 5
   in ALGOL, 95
   in COBOL/COBOL74, 23
   in FORTRAN77, 263
   in PL/I, 181
Entry points, summary of
   in ALGOL, 96
   in COBOL/COBOL74, 24
   in FORTRAN77, 264
   in PL/I, 182

Entry-point declaration
in PL/I, 185
<exception>
as in ALGOLEXCEPTIONNAME, 177, 259
as in ALGOLEXCEPTIONTEXT, 175, 257
as in DBEXCEPTIONNAME, 90
as in DBEXCEPTIONTEXT, 88
as in DBSTATUS, 87
as in FORTRAN77EXCEPTIONNAME, 341
as in FORTRAN77EXCEPTIONTEXT, 339
Exception categories, 90
Exception handling entry points
DBEXCEPTIONNAME, 9
DBEXCEPTIONTEXT, 9
Exception-handling entry points, 9
in ALGOL, 173
in COBOL/COBOL74, 86
in FORTRAN77, 337
in PL/I, 255
Exceptions and errors, 363
<expression>
as in ALGOLDELETE, 131, 215
as in ALGOLFIND, 123, 208
as in ALGOLLOCK, 127, 211
as in DBDELETE, 54
as in DBFIND, 48
as in DBLOCK, 51
as in FORTRAN77DELETE, 296
as in FORTRAN77FIND, 290
as in FORTRAN77LOCK, 293

FIELD data item, 349
as in ALGOLGETDOUBLE, 150, 233
as in ALGOLGETREAL, 144, 227
as in ALGOLPUTDOUBLE, 162, 245
as in ALGOLPUTREAL, 159, 242
as in DBGETDOUBLE, 69
as in DBGETREAL, 65
as in DBPUTDOUBLE, 80
as in DBPUTREAL, 76
as in FORTRAN77GETDOUBLE, 314
as in FORTRAN77GETREAL, 308
as in FORTRAN77PUTDOUBLE, 326
as in FORTRAN77PUTREAL, 323
FIRST
as in ALGOLDELETE, 130, 214
as in ALGOLFIND, 122, 207
as in ALGOLLOCK, 126, 210
as in DBDELETE, 53
as in DBFIND, 47
as in DBLOCK, 50
as in FORTRAN77DELETE, 295

FIRST (cont.)
as in FORTRAN77FIND, 289
as in FORTRAN77LOCK, 292
FORTRAN77 sample programs, See Sample programs
FORTRAN77BEGINTRANSACTION, 274
FORTRAN77CLOSE, 273
FORTRAN77CREATE, 278
FORTRAN77DATA, 334
FORTRAN77DELETE, 294
FORTRAN77ENDTRANSACTION, 276
FORTRAN77EXCEPTIONNAME, 341
FORTRAN77EXCEPTIONTEXT, 339
FORTRAN77FIND, 288
FORTRAN77FREE, 284
FORTRAN77GETCHARACTER, 303
FORTRAN77GETDOUBLE, 314
FORTRAN77GETLOGICAL, 311
FORTRAN77GETREAL, 308
FORTRAN77LOCK, 291
FORTRAN77OPEN, 271
FORTRAN77PUTCHARACTER, 317
FORTRAN77PUTDOUBLE, 326
FORTRAN77PUTLOGICAL, 329
FORTRAN77PUTNULL, 332
FORTRAN77PUTREAL, 323
FORTRAN77RECREATE, 280
FORTRAN77SET, 286
FORTRAN77SETLIMIT, 343
FORTRAN77STATUS, 338
FORTRAN77STORE, 282
FORTRAN77VERB, 297

GET <data type> <structure> <item>
as in ALGOLDATA, 171
as in ALOGLDATA, 253
as in DBDATA, 84
as in FORTRAN77DATA, 335
Glossary, 435
GROUP item, 349
as in ALGOLGETSTRING, 139, 222
as in DBGETDISPLAY, 61
as in FORTRAN77GETCHARACTER, 303

HASH, 349

<in library statement>
for FORTRAN77, 268
<index set>
as in ALGOLFIND, 123, 208
as in ALGOLLOCK, 127, 211

<index set> (cont.)
   as in DBFIND, 47
   as in DBLOCK, 51
   as in FORTRAN77FIND, 290
   as in FORTRAN77LOCK, 293
INQUIRY, 29
   as in ALGOLOPEN, 103, 189
   as in FORTRAN77OPEN, 271
Interpretive Interface, overview, 1
INTLIBERRORS, 9, 86, 173, 255, 337
Invoking entry points
   in ALGOL, 102
   in COBOL/COBOL74, 27
   in FORTRAN77, 270
   in PL/I, 188
ITEM, 348
<item name>, 62
   as in ALGOLGETBOOLEAN, 147, 230
   as in ALGOLGETDOUBLE, 150, 233
   as in ALGOLGETKANJI, 142, 225
   as in ALGOLGETREAL, 144, 227
   as in ALGOLGETSTRING, 139, 222
   as in ALGOLPUTBOOLEAN, 165, 248
   as in ALGOLPUTDOUBLE, 162, 245
   as in ALGOLPUTKANJI, 156, 239
   as in ALGOLPUTNULL, 168, 251
   as in ALGOLPUTREAL, 159, 242
   as in ALGOLPUTSTRING, 153, 236
   as in DBGETBOOLEAN, 67
   as in DBGETDISPLAY, 61
   as in DBGETDOUBLE, 69
   as in DBGETKANJI, 63
   as in DBGETREAL, 65
   as in DBPUTBOOLEAN, 78
   as in DBPUTDISPLAY, 72
   as in DBPUTDOUBLE, 80
   as in DBPUTKANJI, 74
   as in DBPUTNULL, 82
   as in DBPUTREAL, 76
   as in FORTRAN77GETCHARACTER, 303
   as in FORTRAN77GETDOUBLE, 314
   as in FORTRAN77GETKANJI, 306
   as in FORTRAN77GETLOGICAL, 311
   as in FORTRAN77GETREAL, 308
   as in FORTRAN77PUTCHARACTER, 317
   as in FORTRAN77PUTDOUBLE, 326
   as in FORTRAN77PUTKANJI, 320
   as in FORTRAN77PUTLOGICAL, 329
   as in FORTRAN77PUTNULL, 332
   as in FORTRAN77PUTREAL, 323
ITEM-KEY-CLASS, 351
ITEM-OCCURS, 350
ITEM-OWNER-GROUP, 350

ITEM-RECORD-TYPE, 351
ITEM-REQUIRED, 350
ITEM-SCALE-FACTOR, 350
ITEM-SIGNED, 351
ITEM-SIZE, 350
ITEM-SUBSCRIPTS, 349
ITEM-SUBTYPE, 349
ITEM-USAGE, 351

KANJI data item
   as in FORTRAN77DATA, 335
KANJI string, 351
KANJI usage
   as in ALGOLGETKANJI, 142, 225
   as in ALGOLPUTKANJI, 156, 239
   as in DBGETKANJI, 63
   as in DBPUTKANJI, 74
   as in FORTRAN77GETKANJI, 306
   as in FORTRAN77PUTKANJI, 320
KANJI value, 84, 171, 253

LAST
   as in ALGOLDELETE, 130, 214
   as in ALGOLFIND, 122, 207
   as in ALGOLLOCK, 126, 210
   as in DBDELETE, 53
   as in DBFIND, 47
   as in DBLOCK, 50
   as in FORTRAN77DELETE, 295
   as in FORTRAN77FIND, 289
   as in FORTRAN77LOCK, 292
LDBNAME : <data base name> card, 19
<library declaration>, 97, 183
Library declaration
   in ALGOL, 97
   in FORTRAN77, 265
   in PL/I, 183
<library entry point declaration>, 100
   as in PL/I, 186
Library generation, DMINTERPRETER, 13
<library ID>, 97, 183
   for FORTRAN77, 265
Library ID, 25, 100, 183, 186, 268
<library statement>
   for FORTRAN77, 265
<limittype>
   as in ALGOLSETLIMIT, 179, 261
<limitvalue>
   as in ALGOLSETLIMIT, 179, 261
   as in DBSETLIMIT, 93
LINEAR, 349

<link>
   as in ALGOLDELETE, 131, 215
   as in ALGOLFIND, 123, 208
   as in ALGOLLOCK, 127, 211
   as in DBDELETE, 54
   as in DBFIND, 48
   as in DBLOCK, 51
   as in FORTRAN77DELETE, 296
   as in FORTRAN77FIND, 290
   as in FORTRAN77LOCK, 293
LINK, 348
LINK-OCCURS, 352
LINK-TO-DATASET, 351
LOGICAL data item
   as in FORTRAN77DATA, 335
LOGICAL EXPRESSION
   as in FORTRAN77PUTLOGICAL, 329

MAJOR, 351
MINOR, 351

NEXT
   as in ALGOLDELETE, 130, 214
   as in ALGOLFIND, 122, 207
   as in ALGOLLOCK, 126, 210
   as in DBDELETE, 53
   as in DBFIND, 47
   as in DBLOCK, 50
   as in FORTRAN77DELETE, 295
   as in FORTRAN77FIND, 289
   as in FORTRAN77LOCK, 292
NOAUDIT, 32
   as in ALGOLBEGINTRANSACTION, 106, 192
   as in FORTRAN77BEGINTRANSACTION, 274
NONE, 351
NOUPDATE card, 19
NULL data item
   as in FORTRAN77DATA, 335
NULL value, 85, 171, 254
NUMBER data item, 349, 350, 351
   as in ALGOLGETDOUBLE, 150, 233
   as in ALGOLGETREAL, 144, 227
   as in ALGOLPUTDOUBLE, 162, 245
   as in ALGOLPUTREAL, 242
   as in DBGETREAL, 65
   as in DBPUTDOUBLE, 80
   as in DBPUTREAL, 76
   as in FORTRAN77GETDOUBLE, 314
   as in FORTRAN77PUTDOUBLE, 326
   as in FORTRAN77PUTREAL, 323

OCCURS clause, 350, 352
<open disposition>, 29
as in ALGOLOPEN, 103, 189
as in FORTRAN77OPEN, 271

PL/I sample programs, See Sample programs
POPULATION data item, 349
as in ALGOLGETDOUBLE, 150, 233
as in ALGOLGETREAL, 144, 227
as in DBGETDOUBLE, 69
as in DBGETREAL, 65
as in FORTRAN77GETDOUBLE, 314
as in FORTRAN77GETREAL, 308
PRIOR
as in ALGOLDELETE, 130, 214
as in ALGOLFIND, 122, 207
as in ALGOLLOCK, 126, 210
as in DBDELETE, 53
as in DBFIND, 47
as in DBLOCK, 50
as in FORTRAN77DELETE, 295
as in FORTRAN77FIND, 289
as in FORTRAN77LOCK, 292
<procedure heading>, 100
Programming considerations, 393
PUT <data type> <structure> <item>
as in ALGOLDATA, 171, 254
as in DBDATA, 85
as in FORTRAN77DATA, 335

<qualified name>
as in ALGOLDELETE, 131, 215
as in DBDELETE, 54
as in FORTRAN77DELETE, 296
QUEUE : <integer> card, 19

Railroad diagrams, explanation of, 353
REAL data item, 349, 350, 351
as in ALGOLGETDOUBLE, 150, 233
as in ALGOLGETREAL, 144, 227
as in ALGOLPUTDOUBLE, 162, 245
as in ALGOLPUTREAL, 159, 242
as in DBGETDOUBLE, 69
as in DBGETREAL, 65
as in DBPUTDOUBLE, 80
as in DBPUTREAL, 76
as in FORTRAN77DATA, 335
as in FORTRAN77GETDOUBLE, 314
as in FORTRAN77GETREAL, 308
as in FORTRAN77PUTDOUBLE, 326

REAL data item (cont.)
as in FORTRAN77PUTREAL, 323
REAL value, 84, 171, 253
<record statement>
as in ALGOLVERB, 135, 218
as in DBVERB, 57
as in FORTRAN77VERB, 299
<record type>, 36, 38
as in ALGOLCREATE, 110, 196
as in ALGOLRECREATE, 112, 198
as in FORTRAN77CREATE, 278
as in FORTRAN77RECREATE, 280
RECORDTYPE data item
as in ALGOLGETDOUBLE, 150, 233
as in ALGOLGETREAL, 144, 227
as in DBGETDOUBLE, 69
as in DBGETREAL, 65
as in FORTRAN77GETDOUBLE, 314
as in FORTRAN77GETREAL, 308

Sample programs
in ALGOL, 415
in COBOL, 395
in COBOL74, 405
in FORTRAN77, 427
in PL/I, 424
<search statement>
as in ALGOLVERB, 219
as in DBVERB, 58
as in FORTRAN77VERB, 300
<search statment>
as in ALGOLVERB, 136
SET, 348
<set statement>
as in ALGOLVERB, 136, 219
as in DBVERB, 58
as in FORTRAN77VERB, 300
SET-SEARCH-METHOD, 349
SET-SPANS-DATASET, 349
SET-SUBTYPE, 349
Standard entry points, 7
in ALGOL, 103
in COBOL/COBOL74, 29
in FORTRAN77, 271
in PL/I, 189
<status statement>
as in ALGOLVERB, 136, 219
as in FORTRAN77VERB, 300
<string expression>
as in ALGOLPUTSTRING, 153
STRING value, 84, 171, 253

<structure>
as in ALGOLFIND, 121, 206
as in ALGOLFREE, 116, 202
as in ALGOLGETBOOLEAN, 147, 230
as in ALGOLGETDOUBLE, 150, 233
as in ALGOLGETKANJI, 142, 225
as in ALGOLGETREAL, 144, 227
as in ALGOLGETSTRING, 139, 222
as in ALGOLLOCK, 125, 209
as in ALGOLPUTBOOLEAN, 165, 248
as in ALGOLPUTDOUBLE, 162, 245
as in ALGOLPUTKANJI, 156, 239
as in ALGOLPUTNULL, 168, 251
as in ALGOLPUTREAL, 159, 242
as in ALGOLPUTSTRING, 153, 236
as in ALGOLSET, 118, 204
as in ALGOLSTORE, 114, 200
as in DBFIND, 46
as in DBFREE, 42
as in DBGETBOOLEAN, 67
as in DBGETDISPLAY, 61
as in DBGETDOUBLE, 69
as in DBGETKANJI, 63
as in DBGETREAL, 65
as in DBLOCK, 49
as in DBPUTBOOLEAN, 78
as in DBPUTDISPLAY, 72
as in DBPUTDOUBLE, 80
as in DBPUTKANJI, 74
as in DBPUTNULL, 82
as in DBPUTREAL, 76
as in DBSET, 44
as in DBSTORE, 40
as in FORTRAN77FIND, 288
as in FORTRAN77FREE, 284
as in FORTRAN77GETCHARACTER, 303
as in FORTRAN77GETDOUBLE, 314
as in FORTRAN77GETKANJI, 306
as in FORTRAN77GETLOGICAL, 311
as in FORTRAN77GETREAL, 308
as in FORTRAN77LOCK, 291
as in FORTRAN77PUTCHARACTER, 317
as in FORTRAN77PUTDOUBLE, 326
as in FORTRAN77PUTKANJI, 320
as in FORTRAN77PUTLOGICAL, 329
as in FORTRAN77PUTNULL, 332
as in FORTRAN77PUTREAL, 323
as in FORTRAN77SET, 286
as in FORTRAN77STORE, 282
Syntax diagrams for entry point parameters, 377

<target>
   as in ALGOLSET, 118, 204
   as in DBSET, 44
   as in FORTRAN77SET, 286
<text>
   as in ALGOLEXCEPTIONNAME, 177, 259
   as in ALGOLEXCEPTIONTEXT, 175, 257
   as in DBEXCEPTIONNAME, 90
   as in DBEXCEPTIONTEXT, 88
   as in FORTRAN77EXCEPTIONNAME, 341
   as in FORTRAN77EXCEPTIONTEXT, 339
TITLE attribute, 25, 97, 183, 265
TITLE clause, 184, 265
<transaction statement>
   as in ALGOLVERB, 135, 218
   as in DBVERB, 57
   as in FORTRAN77VERB, 299
TYPE data item, 349

UPDATE, 29
   as in ALGOLOPEN, 103, 189
   as in FORTRAN77OPEN, 271

<variable>, 62
   as in ALGOLGETBOOLEAN, 147, 230
   as in ALGOLGETDOUBLE, 150, 233
   as in ALGOLGETKANJI, 142, 225
   as in ALGOLGETREAL, 144, 227
   as in ALGOLGETSTRING, 139, 222
   as in ALGOLPUTKANJI, 156, 239
   as in DBGETBOOLEAN, 67
   as in DBGETDISPLAY, 61
   as in DBGETDOUBLE, 69
   as in DBGETKANJI, 63
   as in DBGETREAL, 65
   as in DBPUTBOOLEAN, 78
   as in DBPUTDISPLAY, 72
   as in DBPUTDOUBLE, 80
   as in DBPUTKANJI, 74
   as in DBPUTREAL, 76
   as in FORTRAN77GETCHARACTER, 303
   as in FORTRAN77GETDOUBLE, 314
   as in FORTRAN77GETKANJI, 306
   as in FORTRAN77GETLOGICAL, 311
   as in FORTRAN77GETREAL, 308

<audit> SYNC
   as in ALGOLENDTRANSACTION, 108, 194
   as in DBENDTRANSACTION, 34
   as in FORTRAN77ENDTRANSACTION, 276

<data set> BEGINNING, 44
as in ALGOLSET, 119, 204
as in FORTRAN77SET, 286
<data set> ENDING, 44
as in ALGOLSET, 119, 204
as in FORTRAN77SET, 286
<index set> <data set>, 45
as in ALGOLSET, 119, 205
as in FORTRAN77SET, 287
<index set> BEGINNING, 45
as in ALGOLSET, 119, 205
as in FORTRAN77SET, 287
<index set> ENDING, 45
as in ALGOLSET, 119, 205
as in FORTRAN77SET, 287
<limittype>=FINDLIMIT
as in ALGOLSETLIMIT, 179, 261
as in DBSETLIMIT, 92

(empty)
as in ALGOLDELETE, 130, 214
as in ALGOLFIND, 122, 207
as in ALGOLLOCK, 126, 210
as in DBLOCK, 50
as in FORTRAN77DELETE, 295
as in FORTRAN77FIND, 289
as in FORTRAN77LOCK, 292

"<blank>", 36, 38
as in ALGOLCREATE, 110, 196
as in ALGOLRECREATE, 113, 199
as in FORTRAN77CREATE, 278
as in FORTRAN77RECREATE, 281
"<integer>", 36, 38
as in ALGOLCREATE, 110, 196
as in ALGOLRECREATE, 112, 198
as in FORTRAN77CREATE, 278
as in FORTRAN77RECREATE, 280